Projektowanie systemów cyfrowych przy użyciu języków wysokiego poziomu ESL

Raport z wykonanych instrukcji

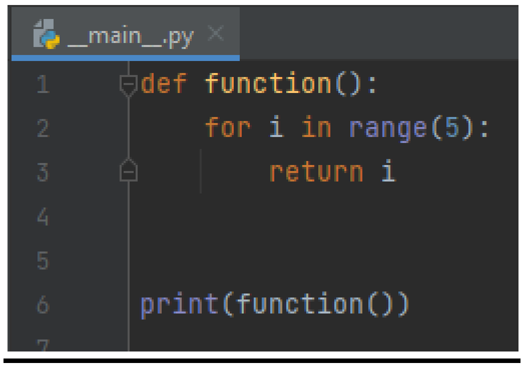
Jarosław Gawlik

Nr albumu: 294139

Elektronika

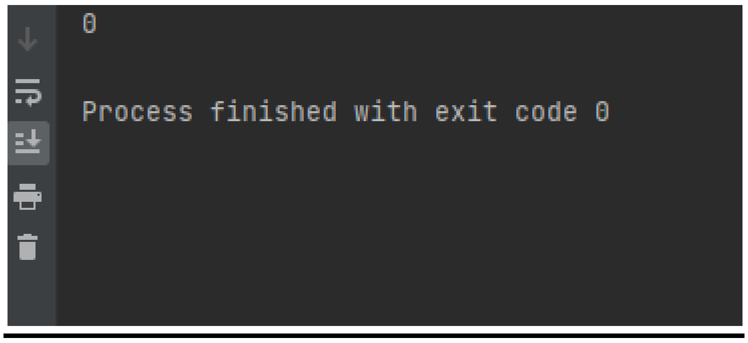
Background information

A small tutorial on generators

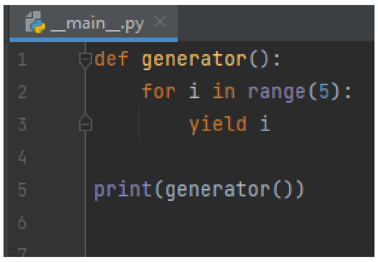


```python
def function():
    for i in range(5):
        return i


print(function())
```

Po uruchomieniu symulacji otrzymano:

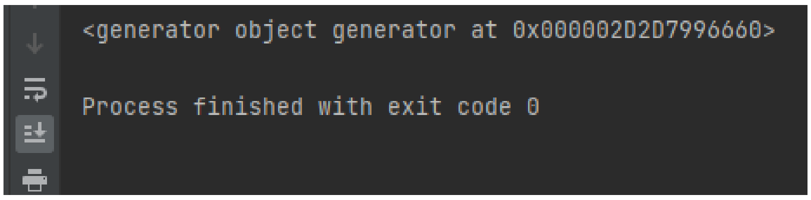


```
0

Process finished with exit code 0
```



```python
def generator():
    for i in range(5):
        yield i

print(generator())
```

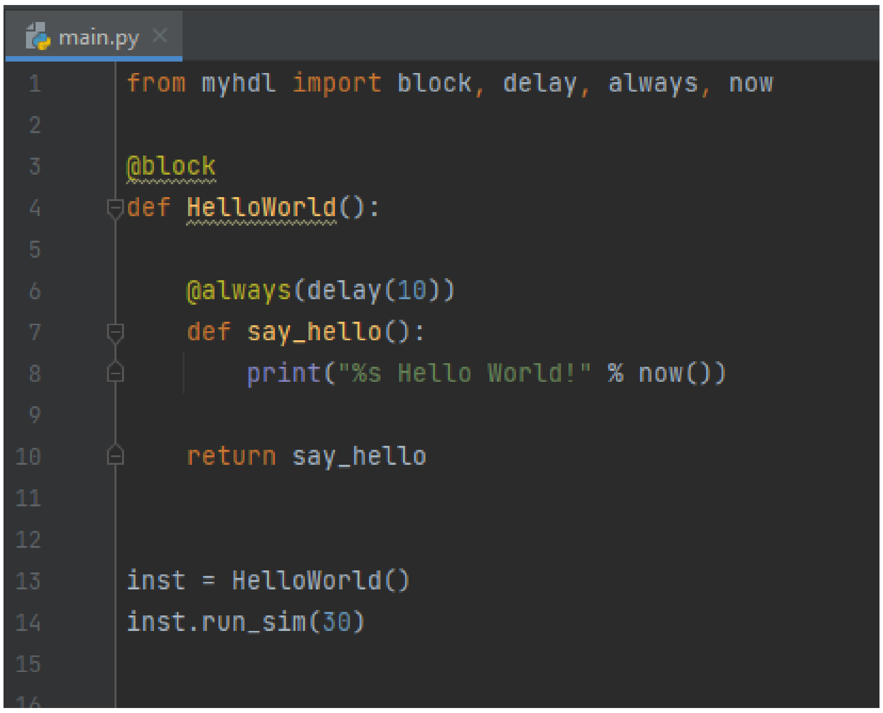
Po uruchomieniu symulacji otrzymano:



```
<generator object generator at 0x000002D2D7996660>

Process finished with exit code 0
```

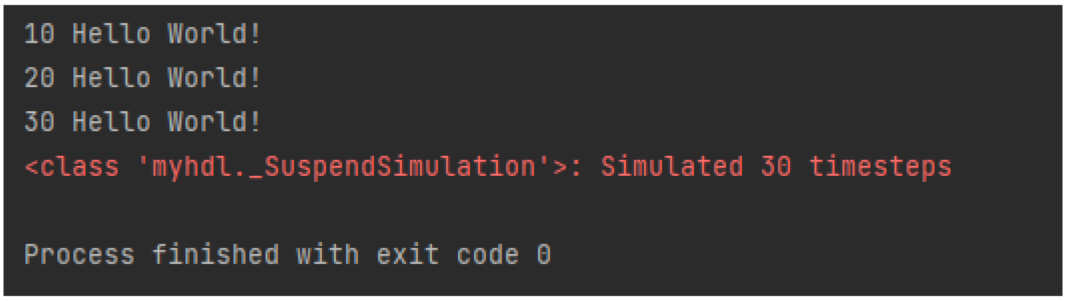
Introduction to MyHDL

A basic MyHDL simulation



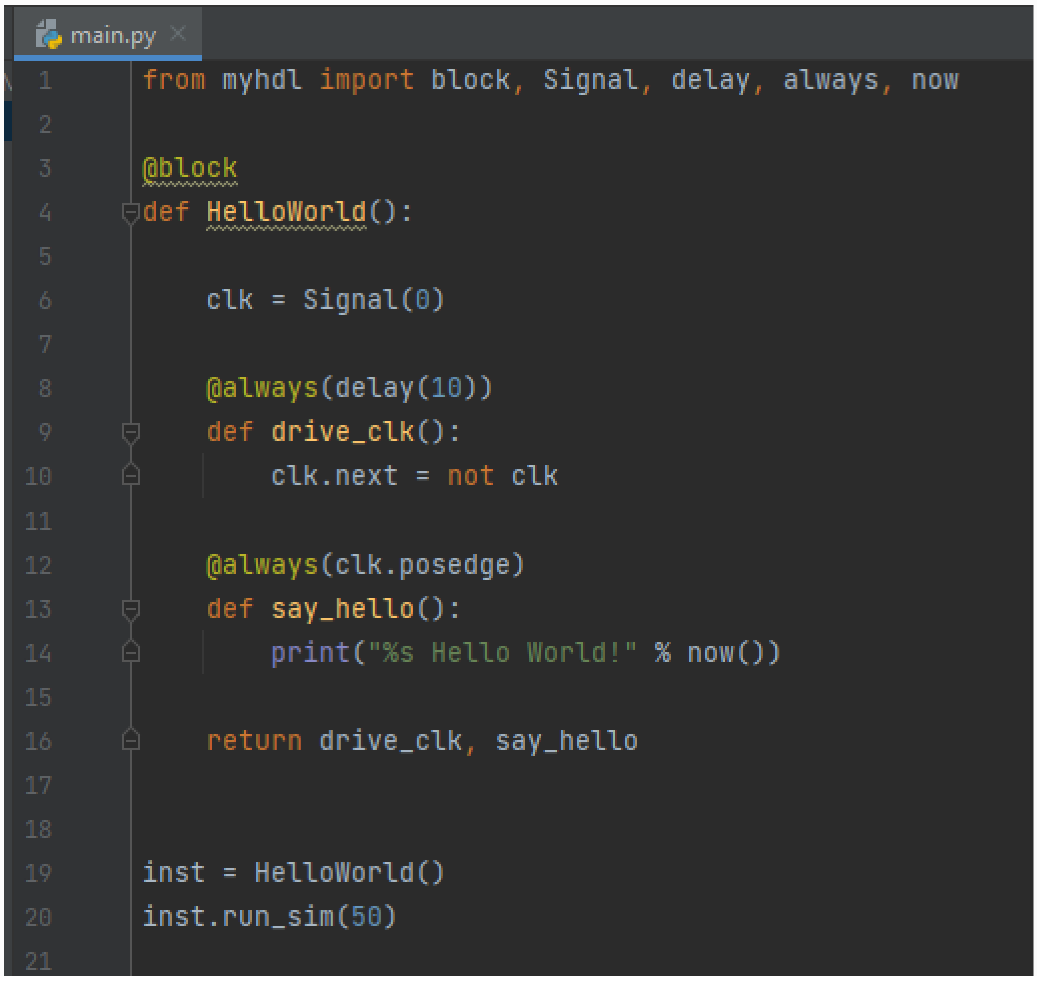
```python
from myhdl import block, delay, always, now

@block
def HelloWorld():

    @always(delay(10))
    def say_hello():
        print("%s Hello World!" % now())

    return say_hello


inst = HelloWorld()
inst.run_sim(30)
```

Po uruchomieniu symulacji otrzymano:



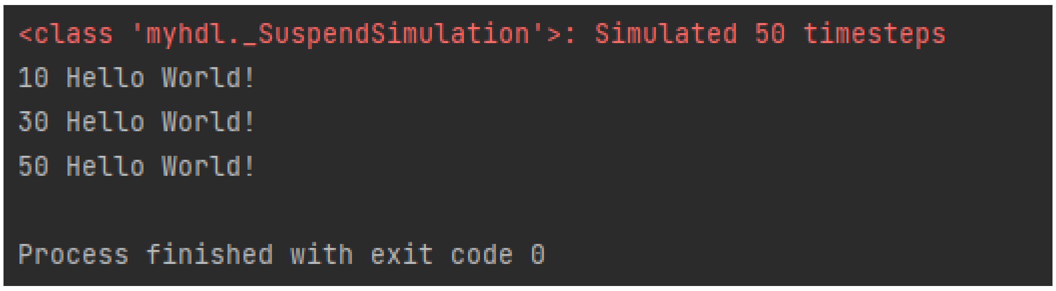
```
10 Hello World!
20 Hello World!
30 Hello World!
<class 'myhdl._SuspendSimulation'>: Simulated 30 timesteps

Process finished with exit code 0
```

Signals and concurrency



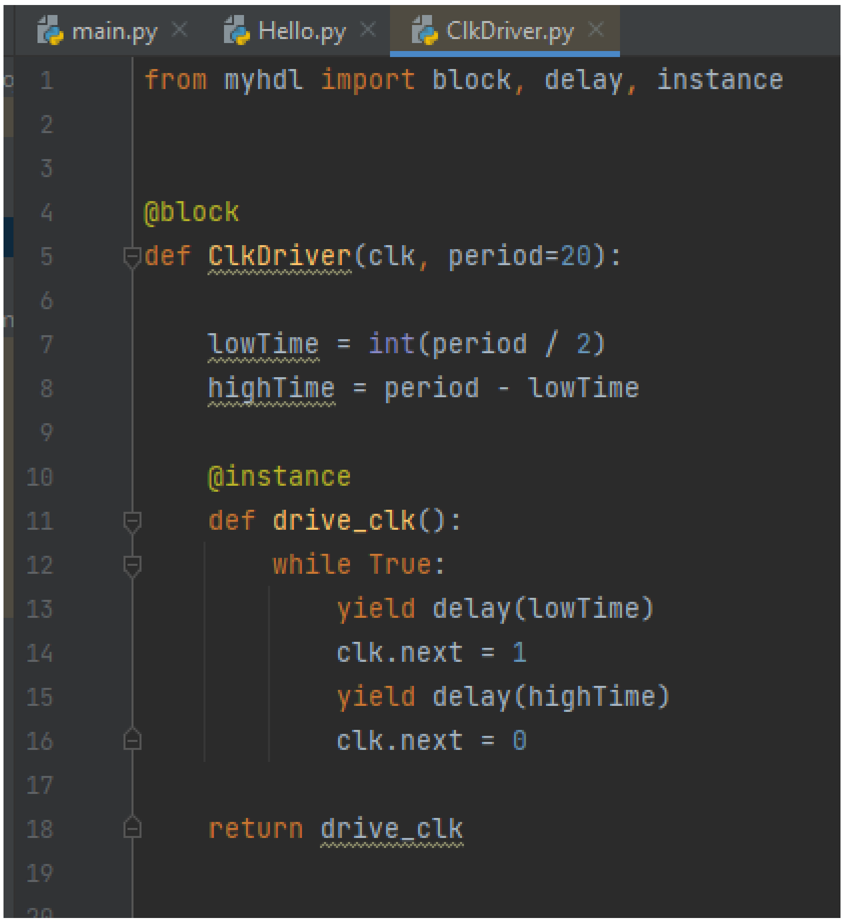
```python
from myhdl import block, Signal, delay, always, now

@block
def HelloWorld():

    clk = Signal(0)

    @always(delay(10))
    def drive_clk():
        clk.next = not clk

    @always(clk.posedge)
    def say_hello():
        print("%s Hello World!" % now())

    return drive_clk, say_hello


inst = HelloWorld()
inst.run_sim(50)

```

Po uruchomieniu symulacji otrzymano:

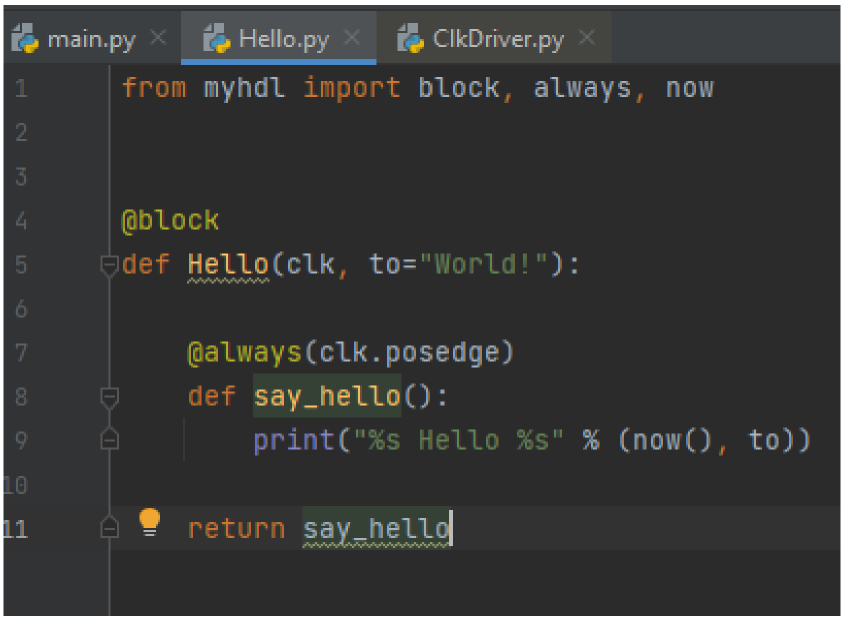


```
<class 'myhdl._SuspendSimulation'>: Simulated 50 timesteps
10 Hello World!
30 Hello World!
50 Hello World!

Process finished with exit code 0
```

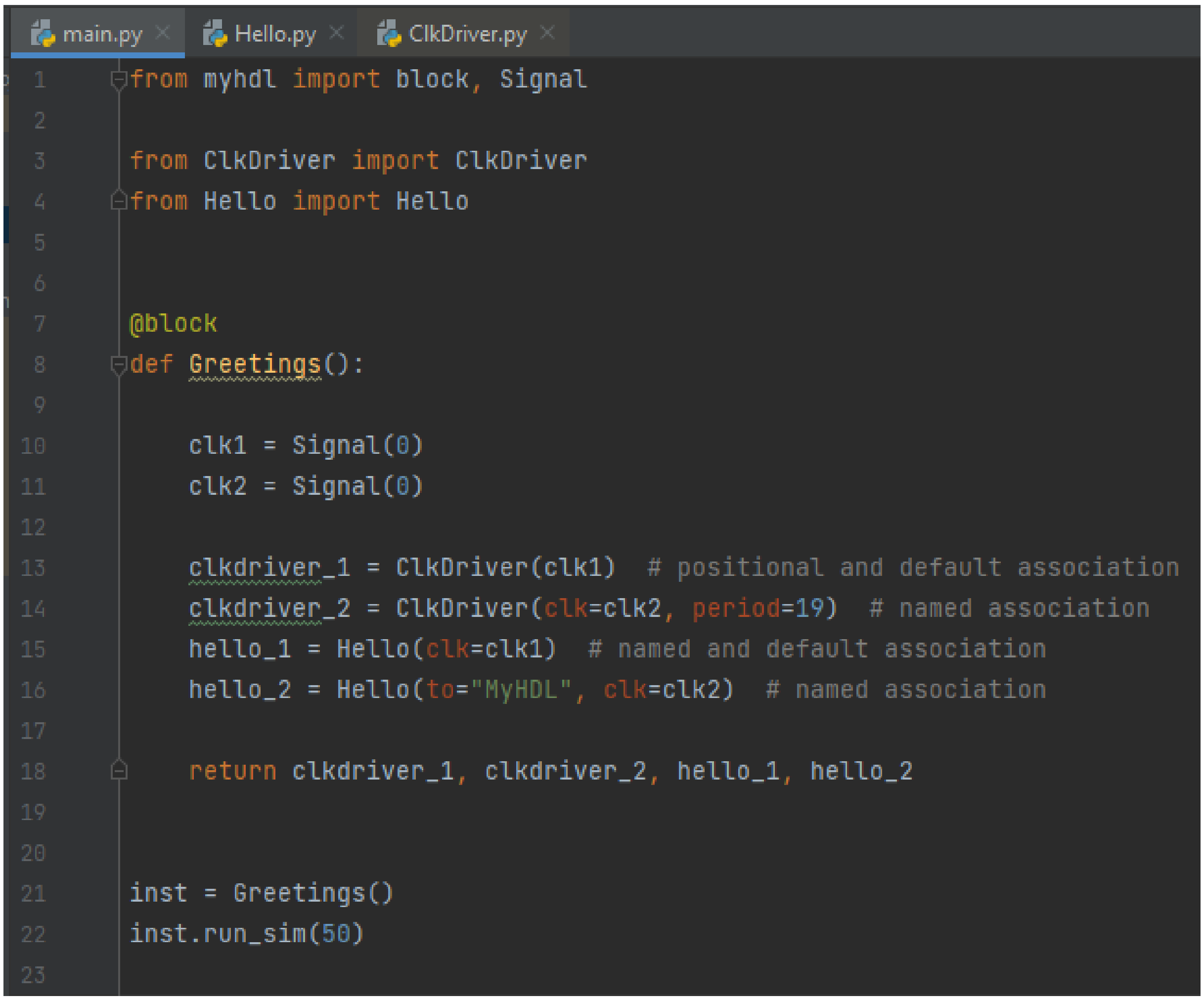
Parameters, ports and hierarchy



```python
from myhdl import block, delay, instance


@block
def ClkDriver(clk, period=20):

    lowTime = int(period / 2)
    highTime = period - lowTime

    @instance
    def drive_clk():
        while True:
            yield delay(lowTime)
            clk.next = 1
            yield delay(highTime)
            clk.next = 0

    return drive_clk

```

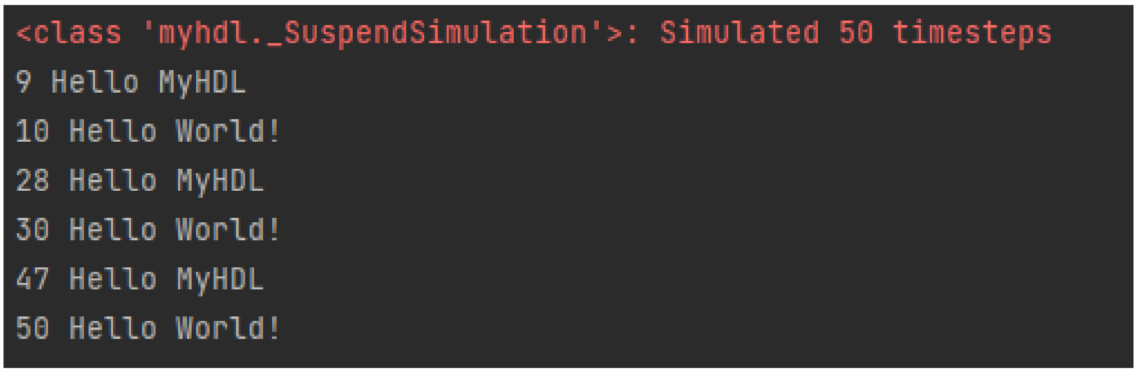


```python
from myhdl import block, always, now


@block
def Hello(clk, to="World!"):

    @always(clk.posedge)
    def say_hello():
        print("%s Hello %s" % (now(), to))

    return say_hello
```



```python
from myhdl import block, Signal

from ClkDriver import ClkDriver
from Hello import Hello


@block
def Greetings():

    clk1 = Signal(0)
    clk2 = Signal(0)

    clkdriver_1 = ClkDriver(clk1)  # positional and default association
    clkdriver_2 = ClkDriver(clk=clk2, period=19)  # named association
    hello_1 = Hello(clk=clk1)  # named and default association
    hello_2 = Hello(to="MyHDL", clk=clk2)  # named association

    return clkdriver_1, clkdriver_2, hello_1, hello_2


inst = Greetings()
inst.run_sim(50)
```

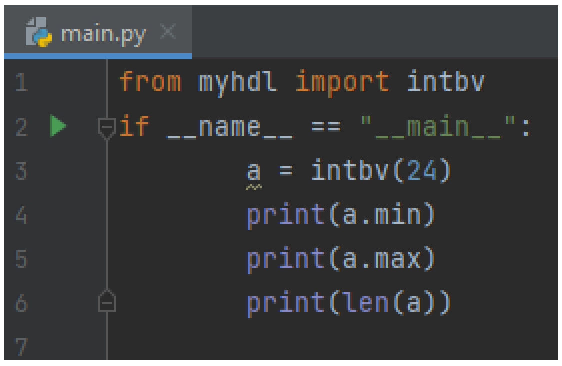
Po uruchomieniu symulacji otrzymano:



```
<class 'myhdl._SuspendSimulation'>: Simulated 50 timesteps
9 Hello MyHDL
10 Hello World!
28 Hello MyHDL
30 Hello World!
47 Hello MyHDL
50 Hello World!
```

Hardware-oriented types

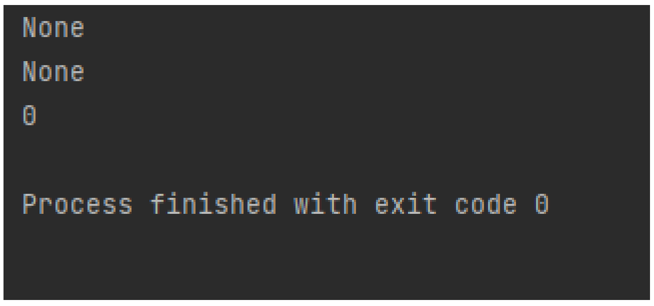
The intbv class



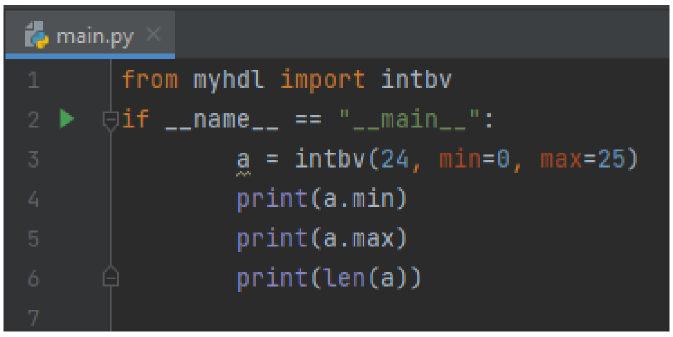
main.py

```python
from myhdl import intbv
if __name__ == "__main__":
        a = intbv(24)
        print(a.min)
        print(a.max)
        print(len(a))
```

Po uruchomieniu symulacji otrzymano:



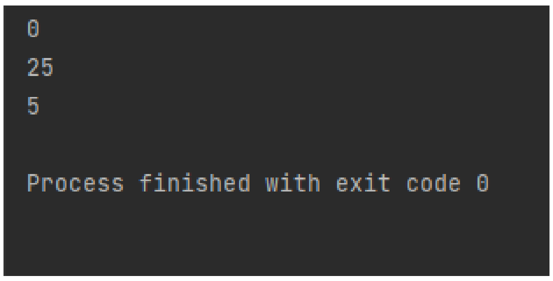
```
None
None
0

Process finished with exit code 0
```



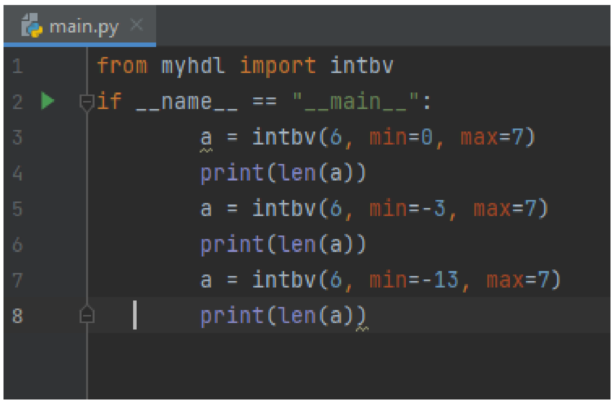
main.py

```python
from myhdl import intbv
if __name__ == "__main__":
        a = intbv(24, min=0, max=25)
        print(a.min)
        print(a.max)
        print(len(a))
```

Po uruchomieniu symulacji otrzymano:

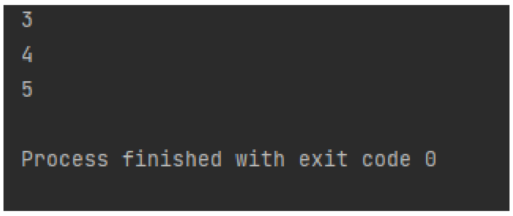


```
0
25
5

Process finished with exit code 0
```



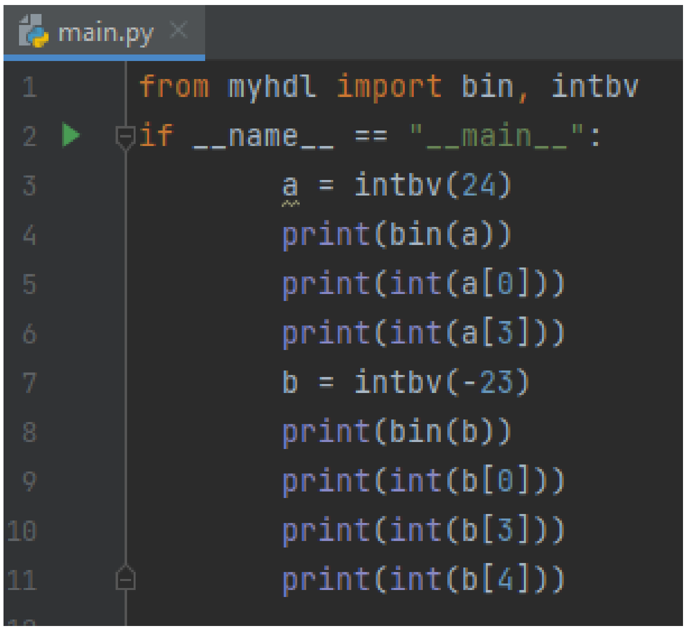
```python
from myhdl import intbv
if __name__ == "__main__":
        a = intbv(6, min=0, max=7)
        print(len(a))
        a = intbv(6, min=-3, max=7)
        print(len(a))
        a = intbv(6, min=-13, max=7)
        print(len(a))
```

Po uruchomieniu symulacji otrzymano:



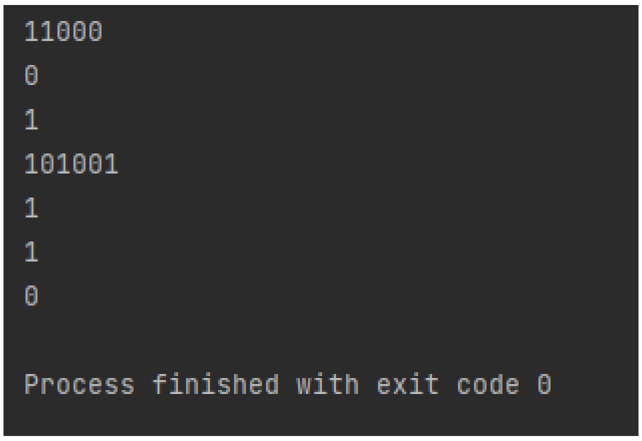
```
3
4
5

Process finished with exit code 0
```

Bit indexing

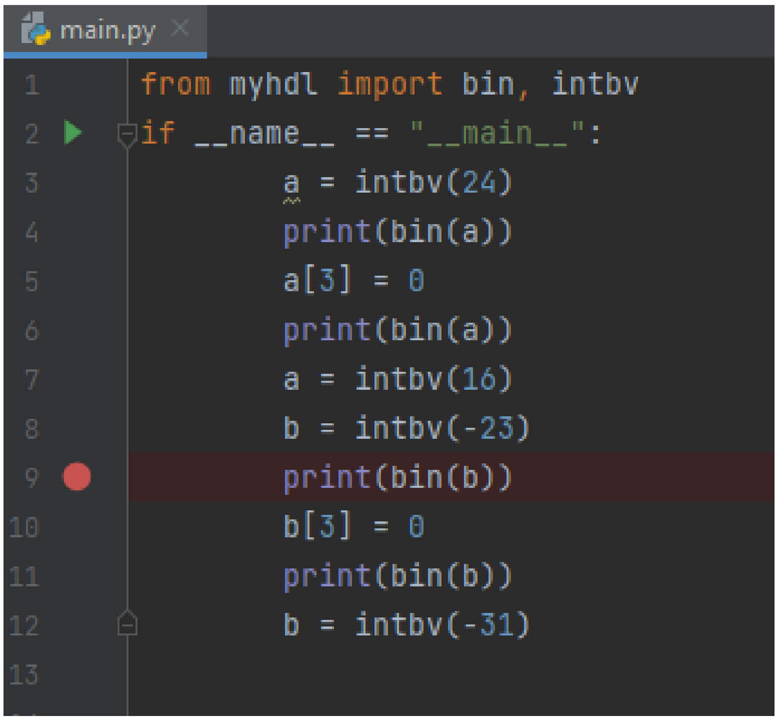


```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(24)
        print(bin(a))
        print(int(a[0]))
        print(int(a[3]))
        b = intbv(-23)
        print(bin(b))
        print(int(b[0]))
        print(int(b[3]))
        print(int(b[4]))
```

Po uruchomieniu symulacji otrzymano:

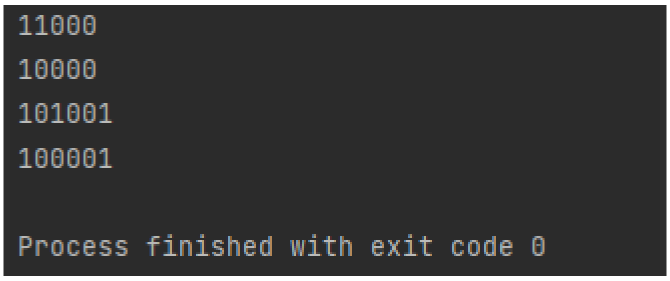


```
11000
0
1
101001
1
1
0

Process finished with exit code 0
```



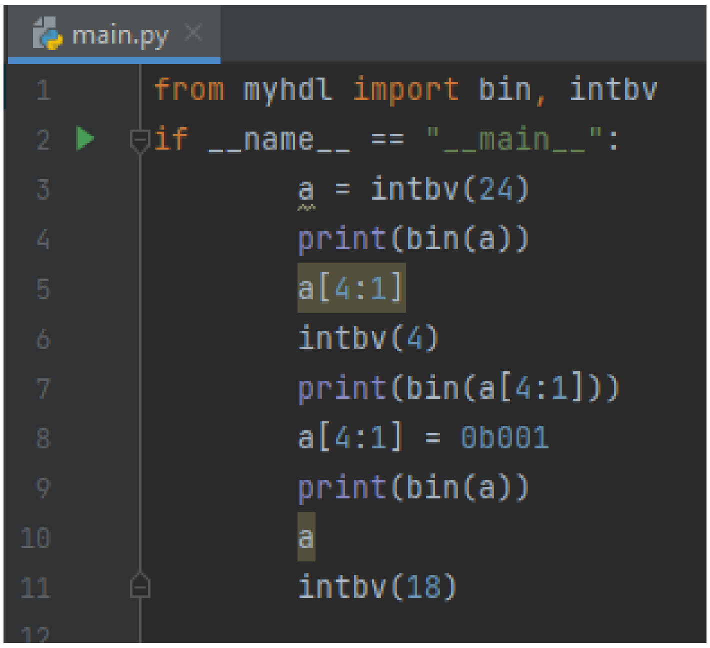
```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(24)
        print(bin(a))
        a[3] = 0
        print(bin(a))
        a = intbv(16)
        b = intbv(-23)
        print(bin(b))
        b[3] = 0
        print(bin(b))
        b = intbv(-31)
```

Po uruchomieniu symulacji otrzymano:



```
11000
10000
101001
100001

Process finished with exit code 0
```

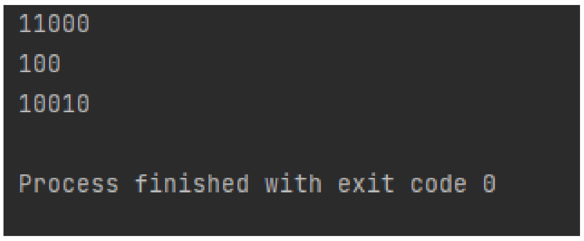
Bit slicing



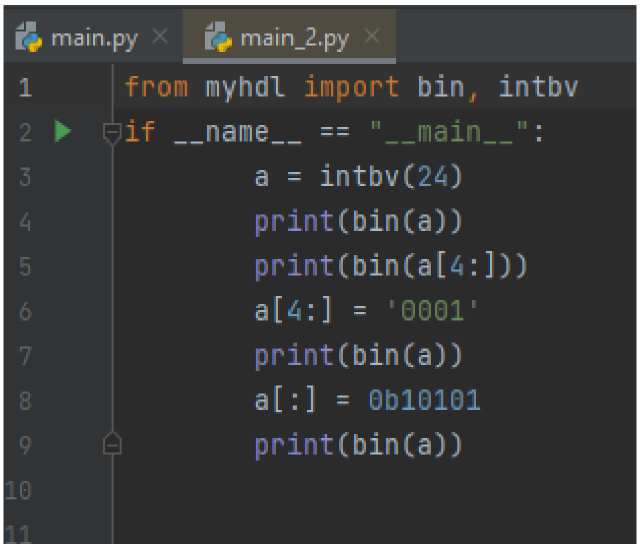
main.py

```
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(24)
        print(bin(a))
        a[4:1]
        intbv(4)
        print(bin(a[4:1]))
        a[4:1] = 0b001
        print(bin(a))
        a
        intbv(18)
```

Po uruchomieniu symulacji otrzymano:



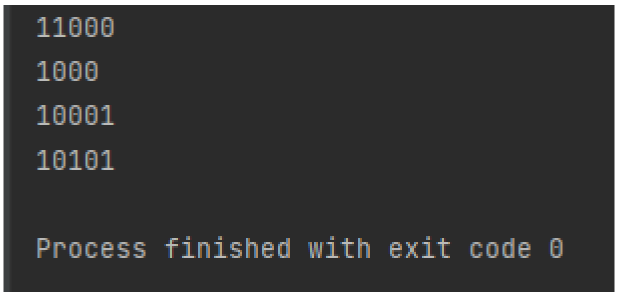
```
11000
100
10010

Process finished with exit code 0
```



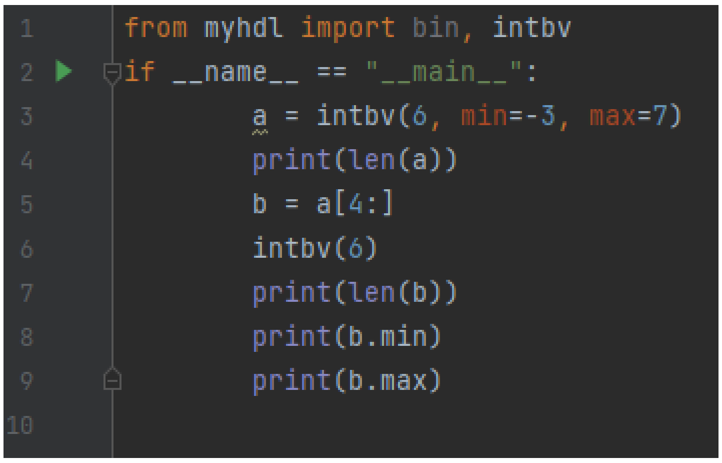
main.py main_2.py

```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(24)
        print(bin(a))
        print(bin(a[4:]))
        a[4:] = '0001'
        print(bin(a))
        a[:] = 0b10101
        print(bin(a))
```

Po uruchomieniu symulacji otrzymano:

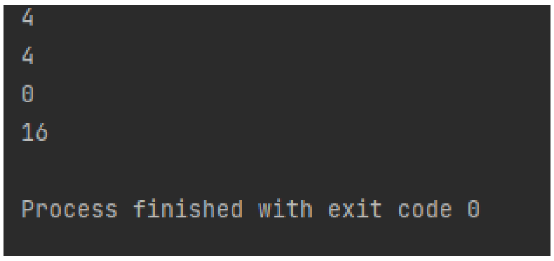


```
11000
1000
10001
10101

Process finished with exit code 0
```

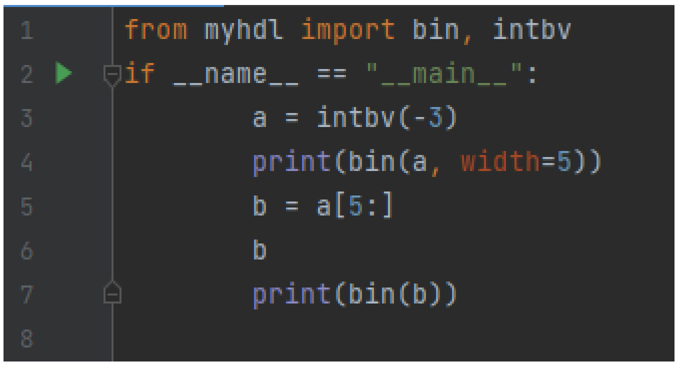


```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(6, min=-3, max=7)
        print(len(a))
        b = a[4:]
        intbv(6)
        print(len(b))
        print(b.min)
        print(b.max)
```

Po uruchomieniu symulacji otrzymano:

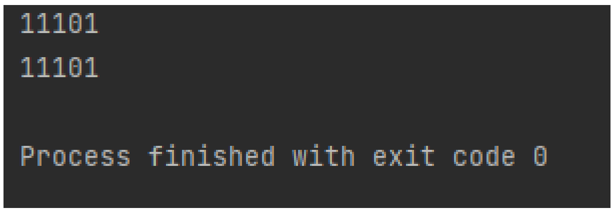


```
4
4
0
16

Process finished with exit code 0
```

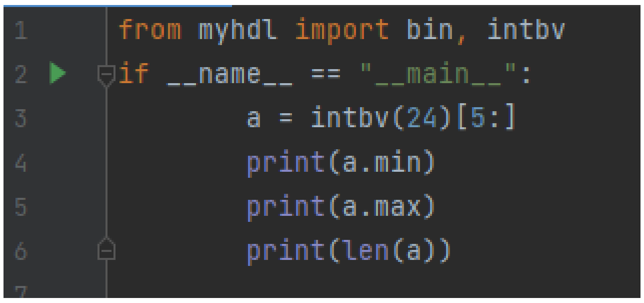


```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(-3)
        print(bin(a, width=5))
        b = a[5:]
        b
        print(bin(b))
```

Po uruchomieniu symulacji otrzymano:

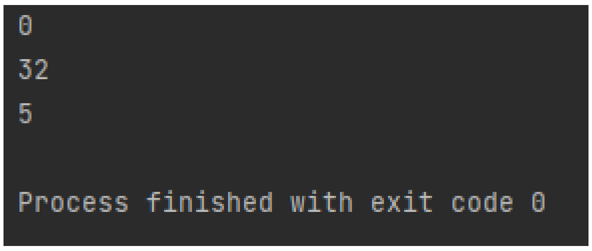


```
11101
11101

Process finished with exit code 0
```



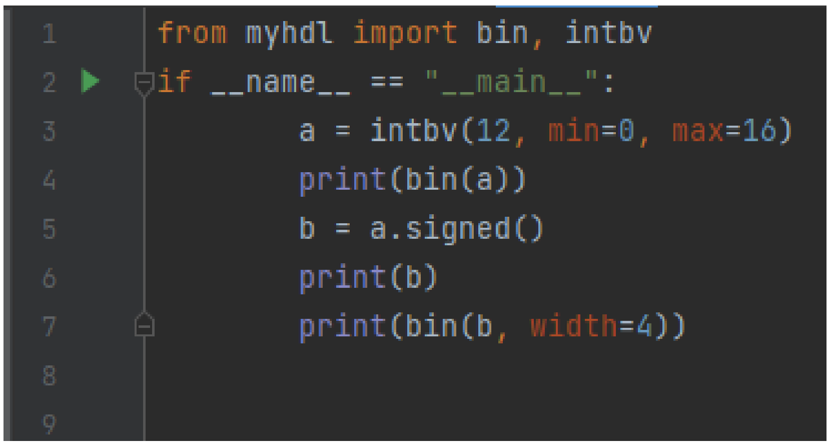
```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(24)[5:]
        print(a.min)
        print(a.max)
        print(len(a))
```

Po uruchomieniu symulacji otrzymano:



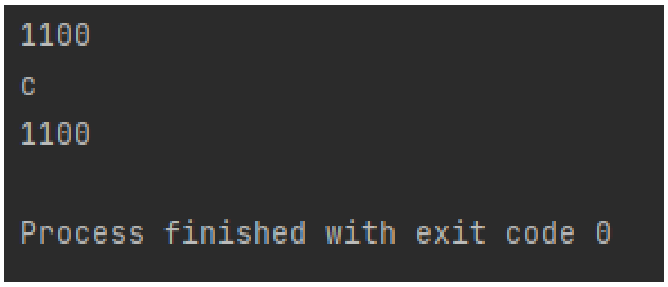
```
0
32
5

Process finished with exit code 0
```

Unsigned and signed representation



```python
from myhdl import bin, intbv
if __name__ == "__main__":
        a = intbv(12, min=0, max=16)
        print(bin(a))
        b = a.signed()
        print(b)
        print(bin(b, width=4))
```

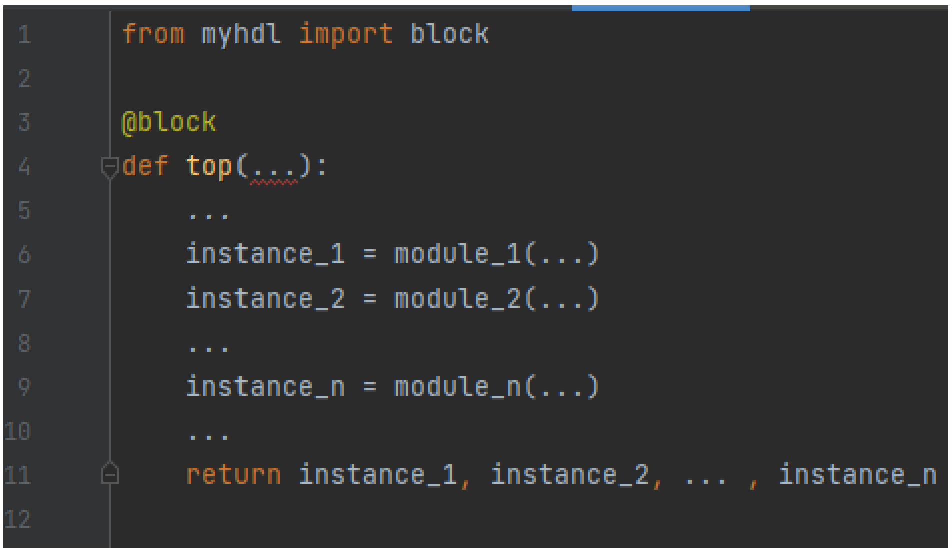
Po uruchomieniu symulacji otrzymano:



```
1100
c
1100

Process finished with exit code 0
```

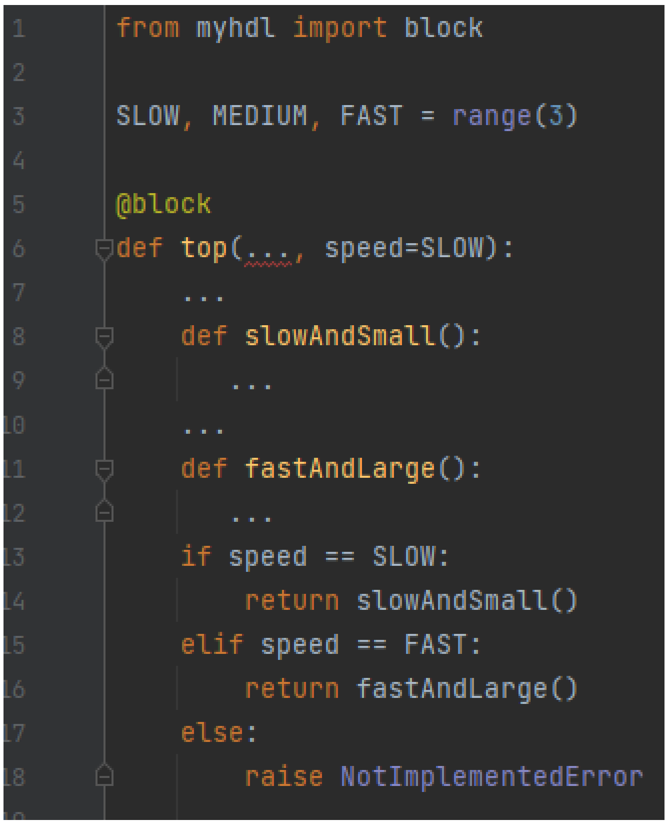
Structural modeling

Introduction



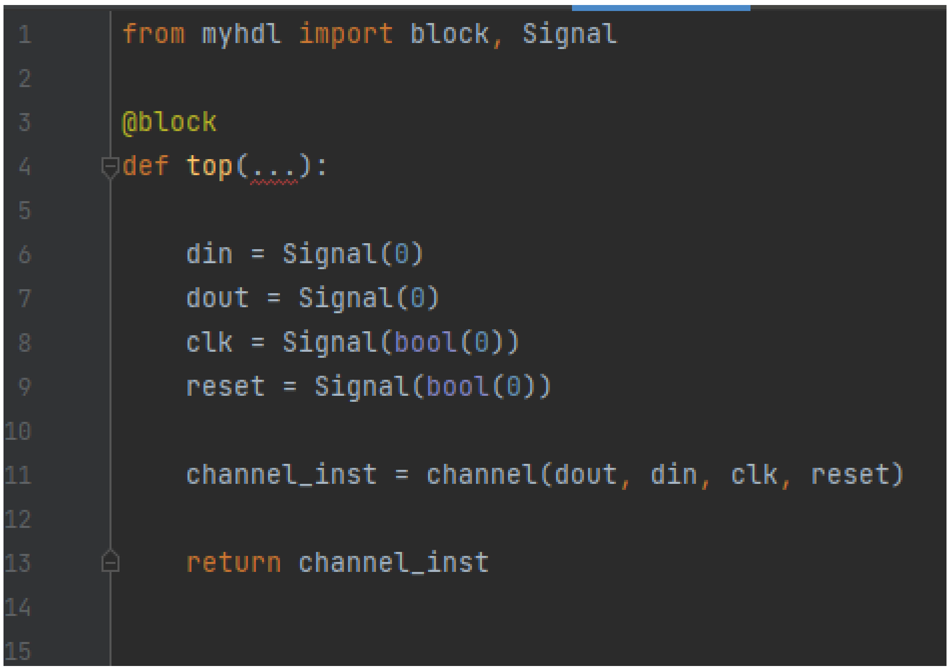
```python
from myhdl import block

@block
def top(...):
    ...
    instance_1 = module_1(...)
    instance_2 = module_2(...)
    ...
    instance_n = module_n(...)
    ...
    return instance_1, instance_2, ... , instance_n

```

Conditional instantiation

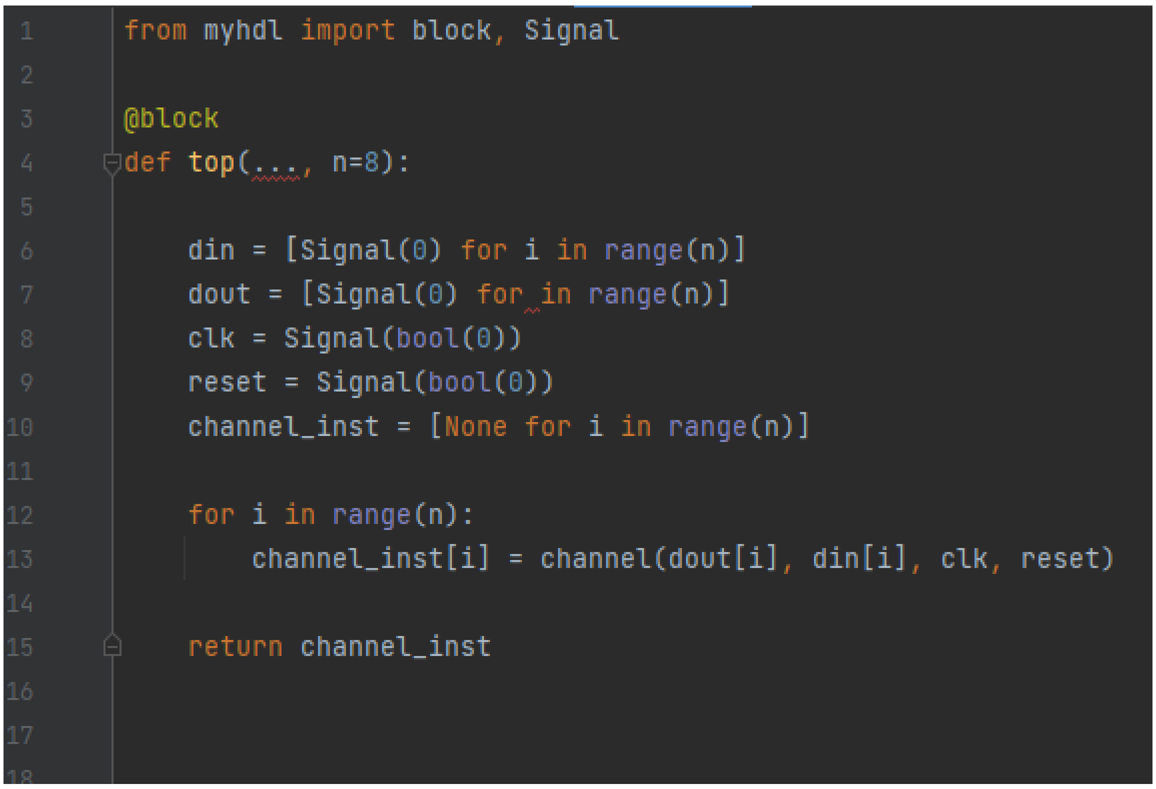


```python
from myhdl import block

SLOW, MEDIUM, FAST = range(3)

@block
def top(..., speed=SLOW):
    ...
    def slowAndSmall():
        ...
    ...
    def fastAndLarge():
        ...
    if speed == SLOW:
        return slowAndSmall()
    elif speed == FAST:
        return fastAndLarge()
    else:
        raise NotImplementedError
```

Lists of instances and signals

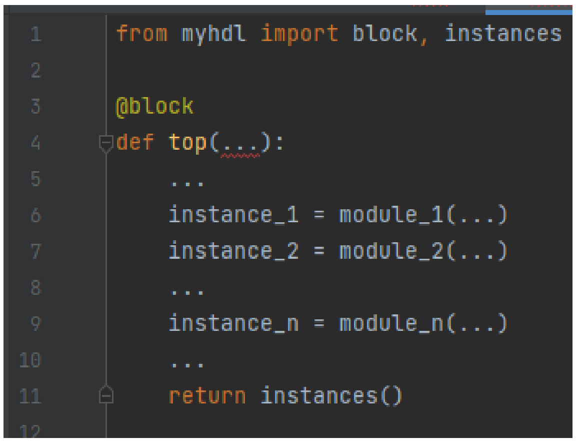


```python
from myhdl import block, Signal

@block
def top(...):

    din = Signal(0)
    dout = Signal(0)
    clk = Signal(bool(0))
    reset = Signal(bool(0))

    channel_inst = channel(dout, din, clk, reset)

    return channel_inst


```



```python
from myhdl import block, Signal

@block
def top(..., n=8):

    din = [Signal(0) for i in range(n)]
    dout = [Signal(0) for in range(n)]
    clk = Signal(bool(0))
    reset = Signal(bool(0))
    channel_inst = [None for i in range(n)]

    for i in range(n):
        channel_inst[i] = channel(dout[i], din[i], clk, reset)

    return channel_inst
```

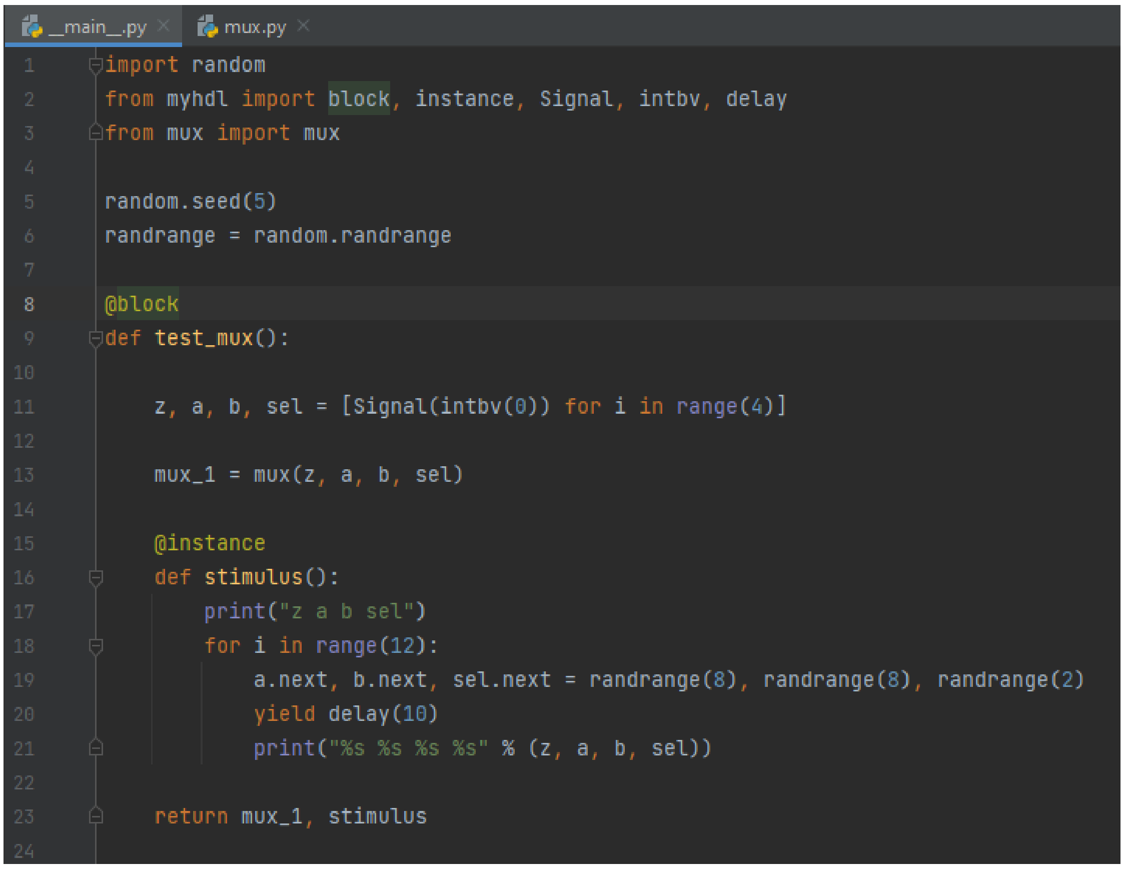
Inferring the list of instances



```python
from myhdl import block, instances

@block
def top(...):
    ...
    instance_1 = module_1(...)
    instance_2 = module_2(...)
    ...
    instance_n = module_n(...)
    ...
    return instances()
```

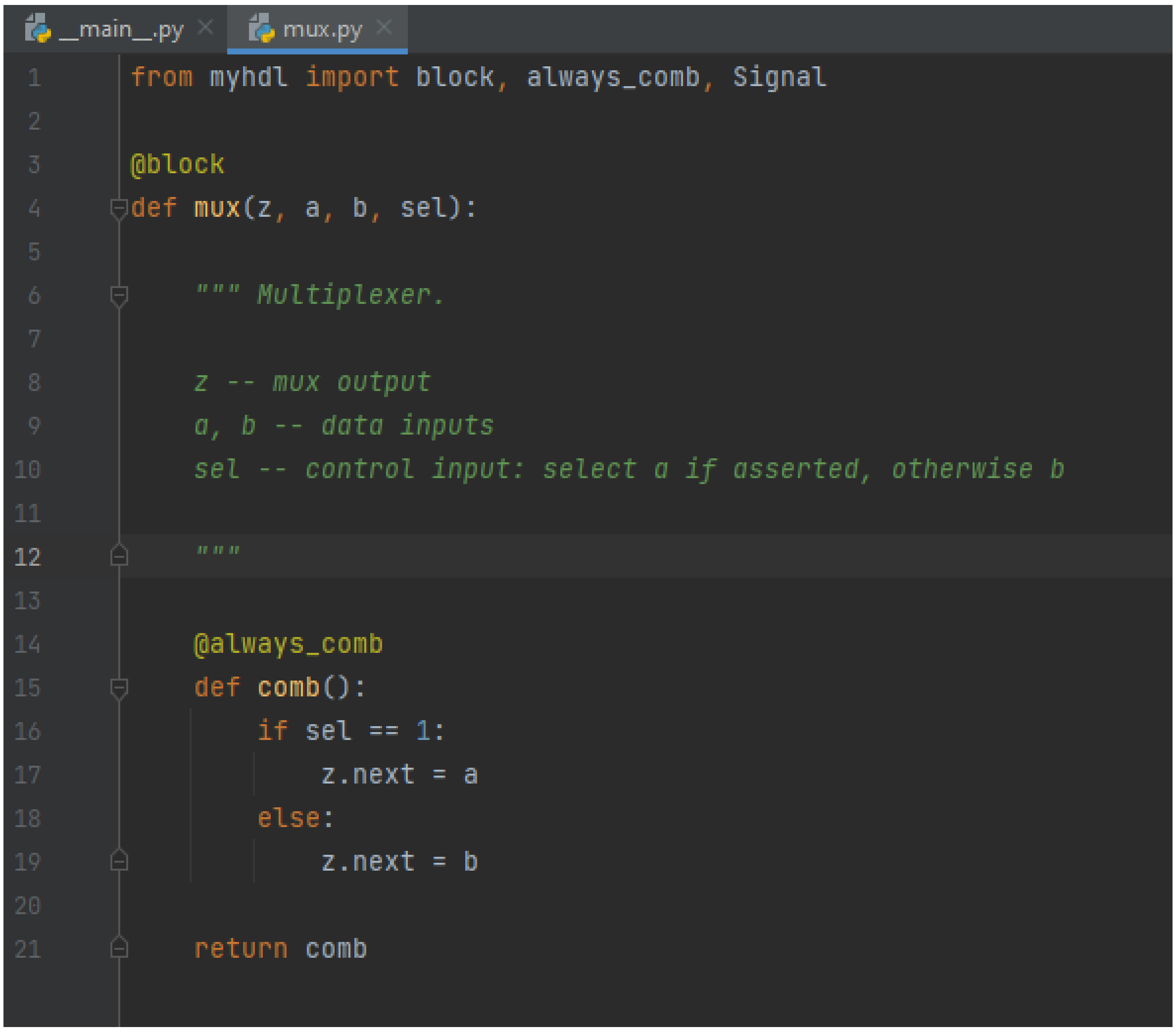
RTL modeling

Combinatorial logic



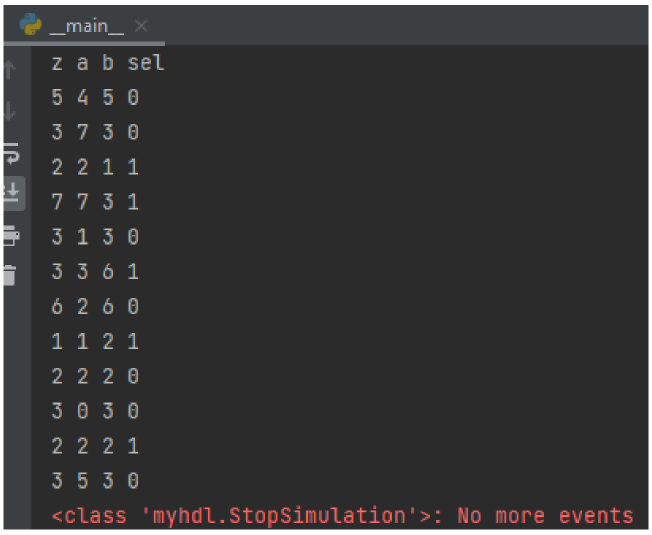
__main__.py mux.py

```python
import random
from myhdl import block, instance, Signal, intbv, delay
from mux import mux

random.seed(5)
randrange = random.randrange

@block
def test_mux():

    z, a, b, sel = [Signal(intbv(0)) for i in range(4)]

    mux_1 = mux(z, a, b, sel)

    @instance
    def stimulus():
        print("z a b sel")
        for i in range(12):
            a.next, b.next, sel.next = randrange(8), randrange(8), randrange(2)
            yield delay(10)
            print("%s %s %s %s" % (z, a, b, sel))

    return mux_1, stimulus
```



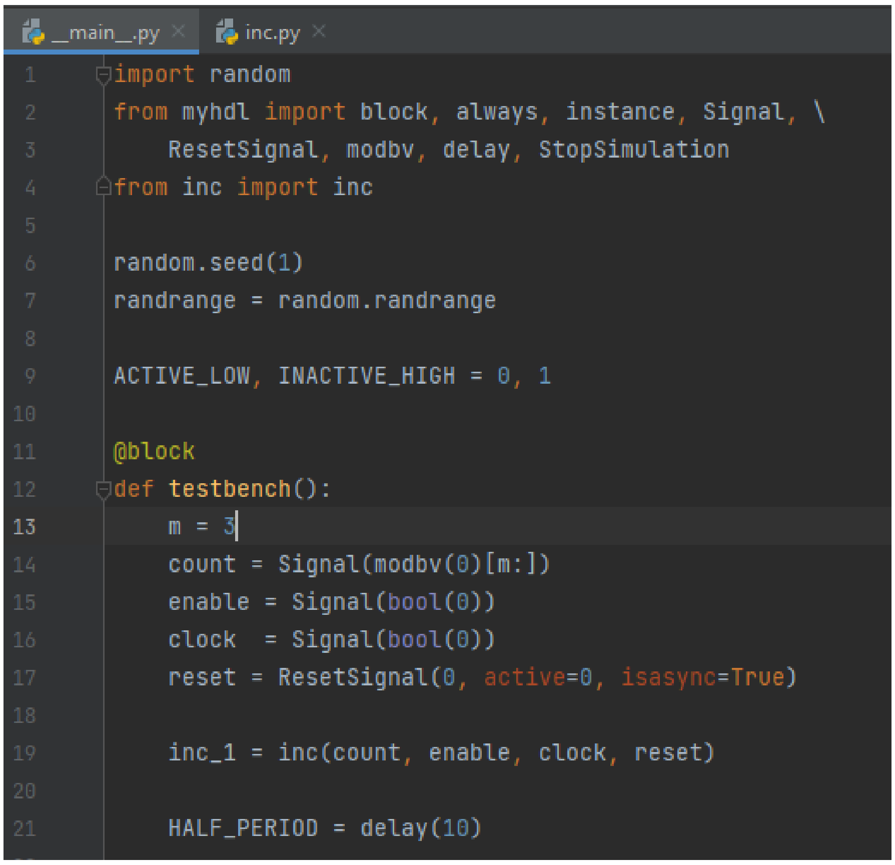
```
from myhdl import block, always_comb, Signal

@block
def mux(z, a, b, sel):

    """ Multiplexer.

    z -- mux output
    a, b -- data inputs
    sel -- control input: select a if asserted, otherwise b

    """

    @always_comb
    def comb():
        if sel == 1:
            z.next = a
        else:
            z.next = b

    return comb
```

Po uruchomieniu symulacji otrzymano:

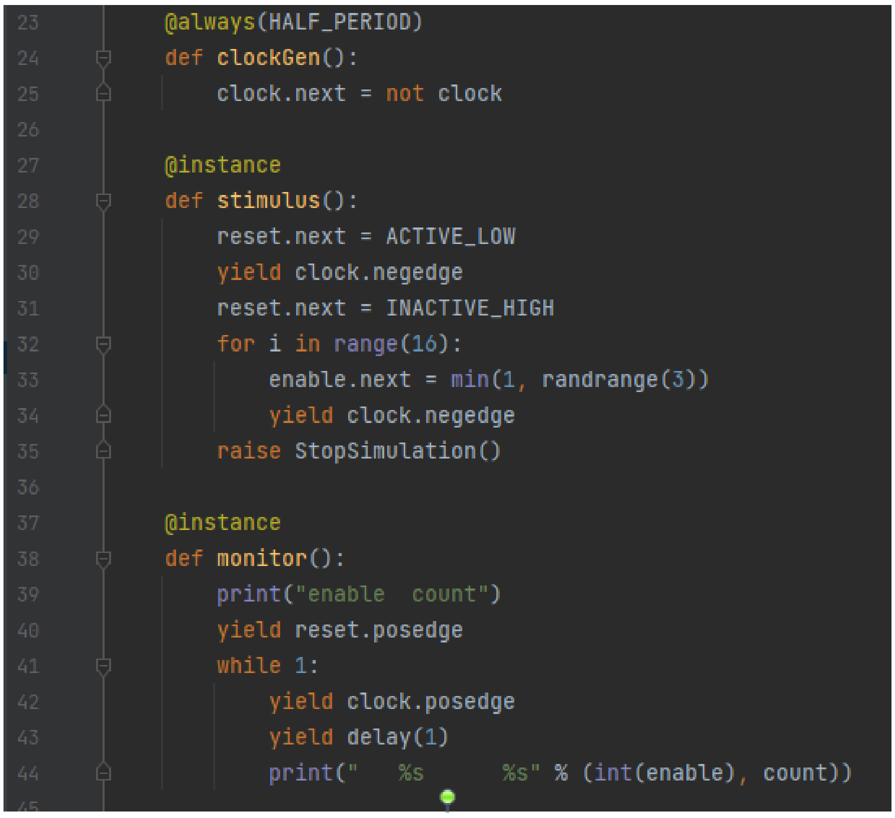


```
z a b sel
5 4 5 0
3 7 3 0
2 2 1 1
7 7 3 1
3 1 3 0
3 3 6 1
6 2 6 0
1 1 2 1
2 2 2 0
3 0 3 0
2 2 2 1
3 5 3 0
<class 'myhdl.StopSimulation'>: No more events
```

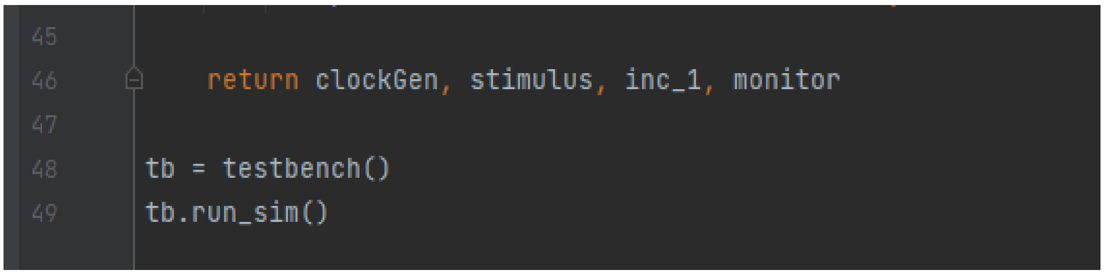
Sequential logic



```python
import random
from myhdl import block, always, instance, Signal, \
    ResetSignal, modbv, delay, StopSimulation
from inc import inc

random.seed(1)
randrange = random.randrange

ACTIVE_LOW, INACTIVE_HIGH = 0, 1

@block
def testbench():
    m = 3
    count = Signal(modbv(0)[m:])
    enable = Signal(bool(0))
    clock  = Signal(bool(0))
    reset = ResetSignal(0, active=0, isasync=True)

    inc_1 = inc(count, enable, clock, reset)

    HALF_PERIOD = delay(10)
```

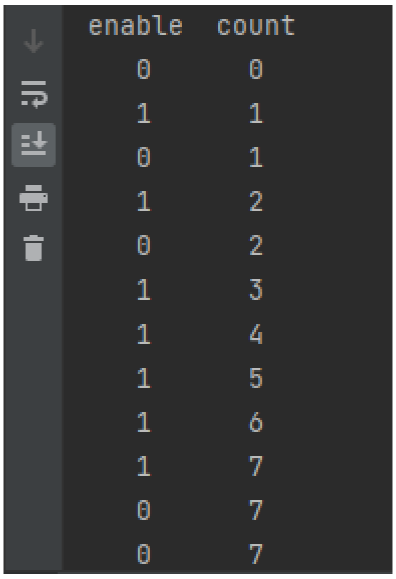


```python
    @always(HALF_PERIOD)
    def clockGen():
        clock.next = not clock

    @instance
    def stimulus():
        reset.next = ACTIVE_LOW
        yield clock.negedge
        reset.next = INACTIVE_HIGH
        for i in range(16):
            enable.next = min(1, randrange(3))
            yield clock.negedge
        raise StopSimulation()

    @instance
    def monitor():
        print("enable  count")
        yield reset.posedge
        while 1:
            yield clock.posedge
            yield delay(1)
            print("   %s      %s" % (int(enable), count))
```

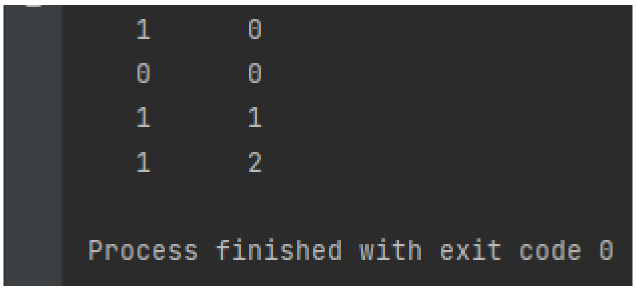


```
    return clockGen, stimulus, inc_1, monitor

tb = testbench()
tb.run_sim()
```

Po uruchomieniu symulacji otrzymano:

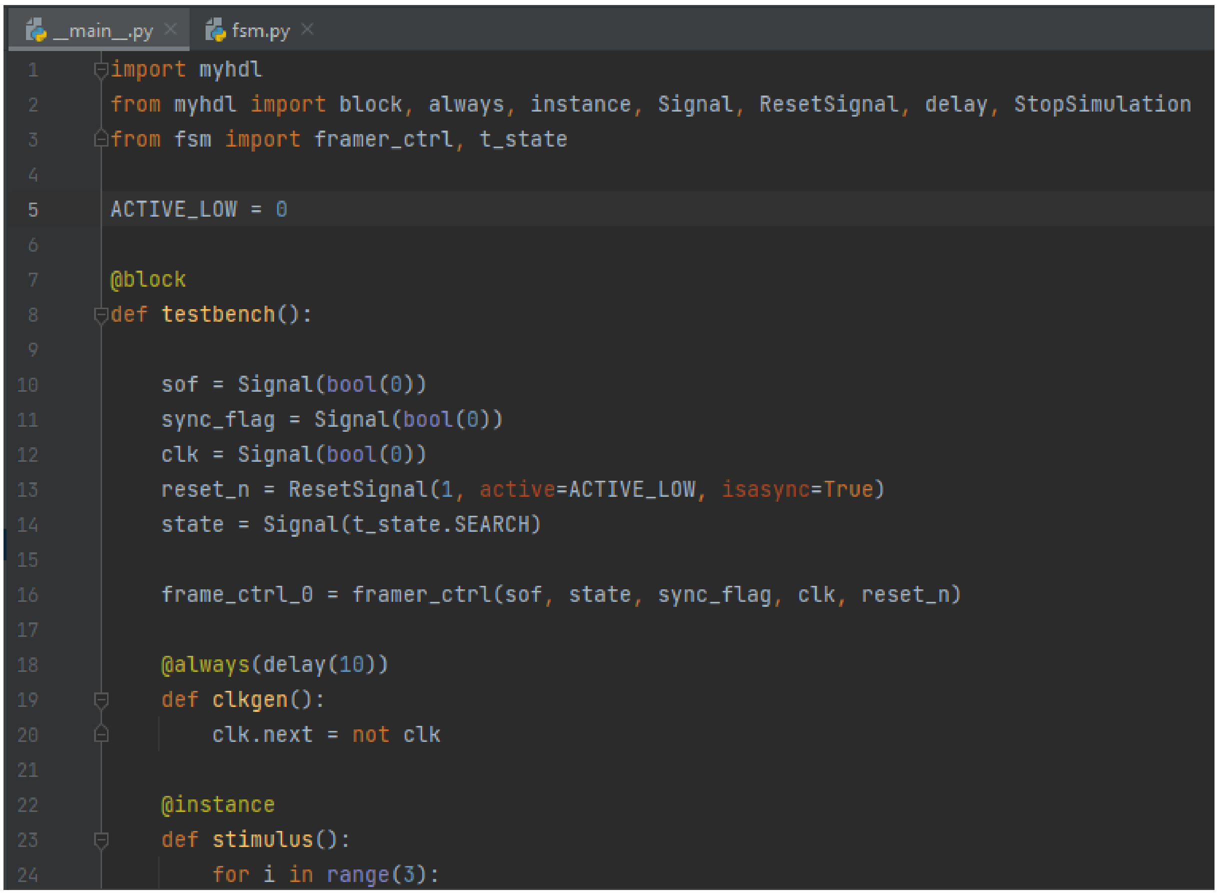


```
enable  count
   0      0
   1      1
   0      1
   1      2
   0      2
   1      3
   1      4
   1      5
   1      6
   1      7
   0      7
   0      7
```

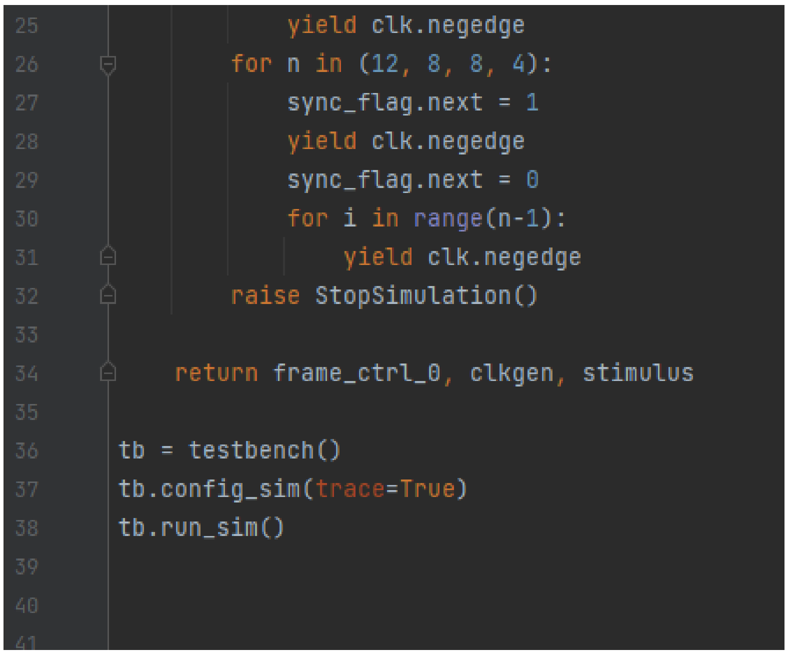


```
   1      0
   0      0
   1      1
   1      2

Process finished with exit code 0
```

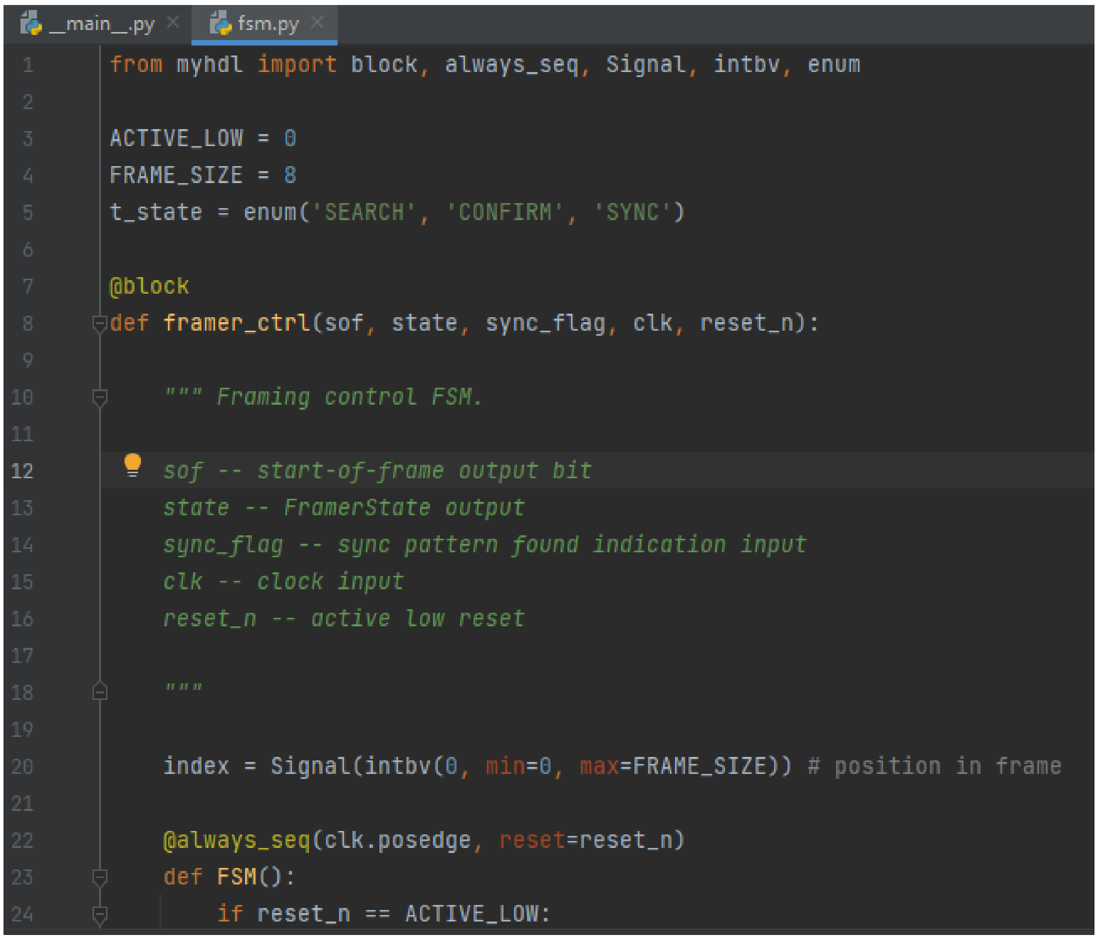
Finite State Machine modeling



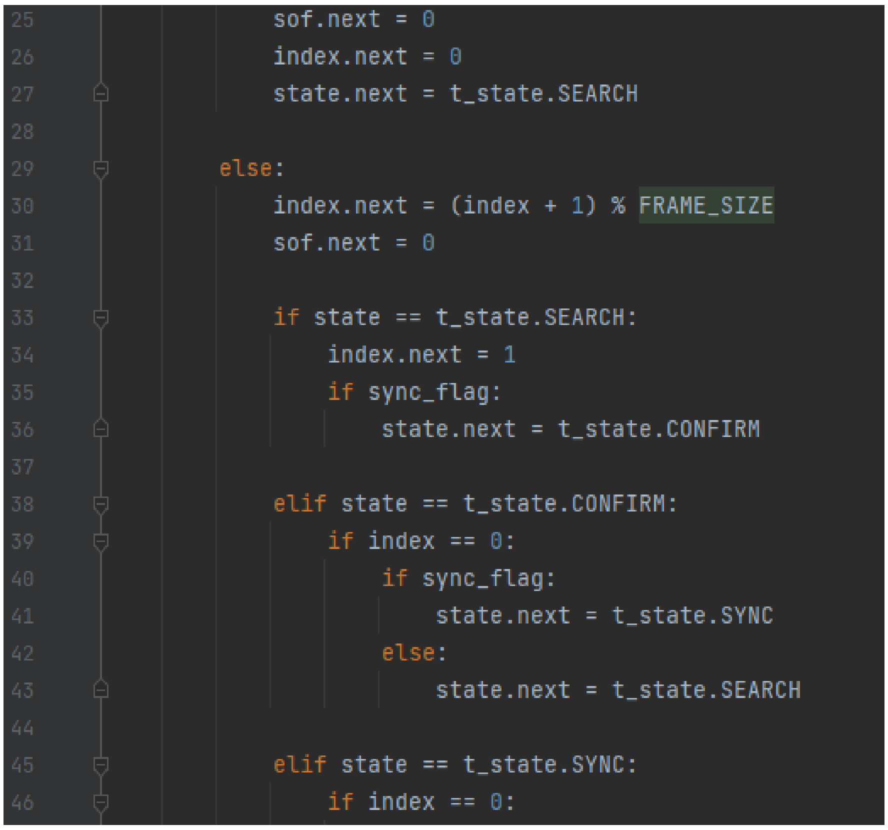
```python
import myhdl
from myhdl import block, always, instance, Signal, ResetSignal, delay, StopSimulation
from fsm import framer_ctrl, t_state

ACTIVE_LOW = 0

@block
def testbench():

    sof = Signal(bool(0))
    sync_flag = Signal(bool(0))
    clk = Signal(bool(0))
    reset_n = ResetSignal(1, active=ACTIVE_LOW, isasync=True)
    state = Signal(t_state.SEARCH)

    frame_ctrl_0 = framer_ctrl(sof, state, sync_flag, clk, reset_n)

    @always(delay(10))
    def clkgen():
        clk.next = not clk

    @instance
    def stimulus():
        for i in range(3):
```



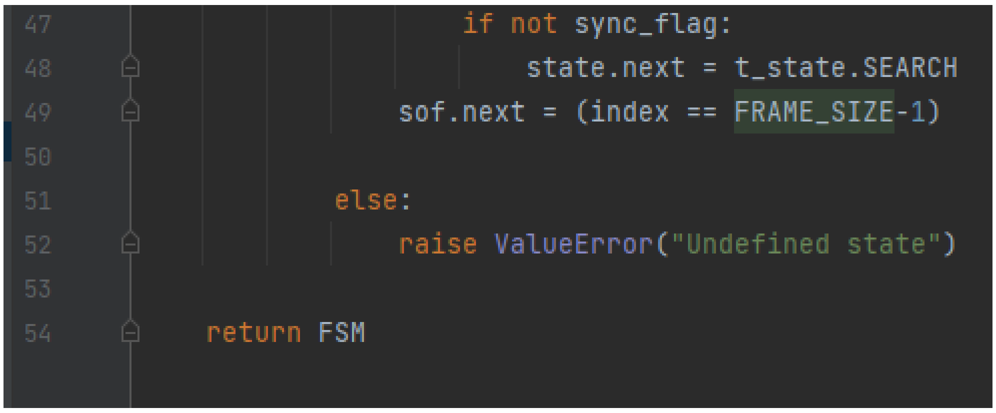
```python
            yield clk.negedge
        for n in (12, 8, 8, 4):
            sync_flag.next = 1
            yield clk.negedge
            sync_flag.next = 0
            for i in range(n-1):
                yield clk.negedge
        raise StopSimulation()

    return frame_ctrl_0, clkgen, stimulus

tb = testbench()
tb.config_sim(trace=True)
tb.run_sim()
```



```python
from myhdl import block, always_seq, Signal, intbv, enum

ACTIVE_LOW = 0
FRAME_SIZE = 8
t_state = enum('SEARCH', 'CONFIRM', 'SYNC')

@block
def framer_ctrl(sof, state, sync_flag, clk, reset_n):

    """ Framing control FSM.

    sof -- start-of-frame output bit
    state -- FramerState output
    sync_flag -- sync pattern found indication input
    clk -- clock input
    reset_n -- active low reset

    """

    index = Signal(intbv(0, min=0, max=FRAME_SIZE)) # position in frame

    @always_seq(clk.posedge, reset=reset_n)
    def FSM():
        if reset_n == ACTIVE_LOW:
```

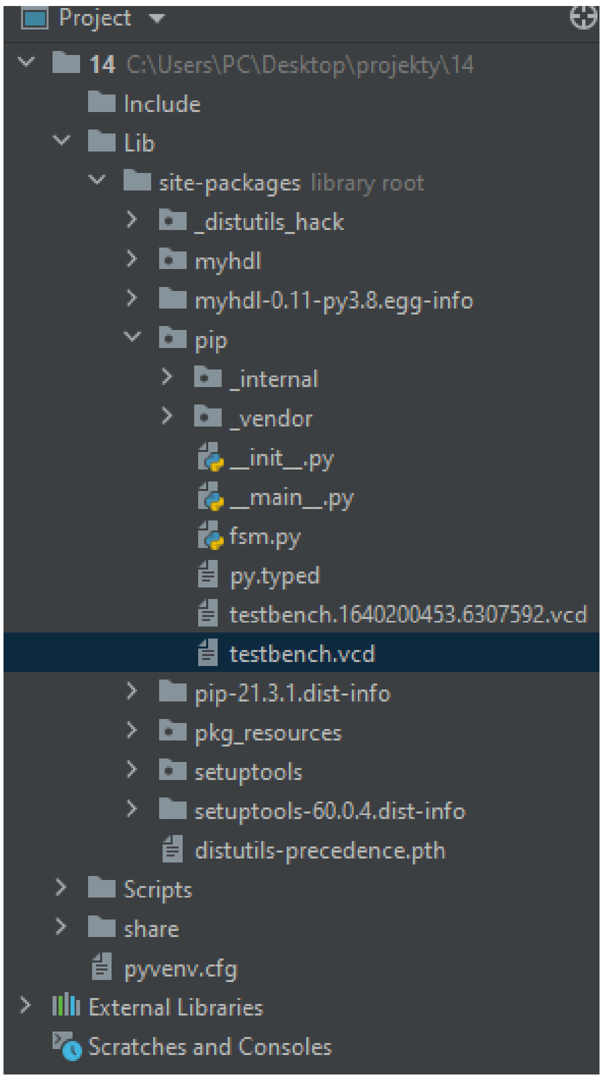


```python
            sof.next = 0
            index.next = 0
            state.next = t_state.SEARCH

        else:
            index.next = (index + 1) % FRAME_SIZE
            sof.next = 0

            if state == t_state.SEARCH:
                index.next = 1
                if sync_flag:
                    state.next = t_state.CONFIRM

            elif state == t_state.CONFIRM:
                if index == 0:
                    if sync_flag:
                        state.next = t_state.SYNC
                    else:
                        state.next = t_state.SEARCH

            elif state == t_state.SYNC:
                if index == 0:
```



```
                if not sync_flag:
                    state.next = t_state.SEARCH
                sof.next = (index == FRAME_SIZE-1)

            else:
                raise ValueError("Undefined state")

    return FSM
```

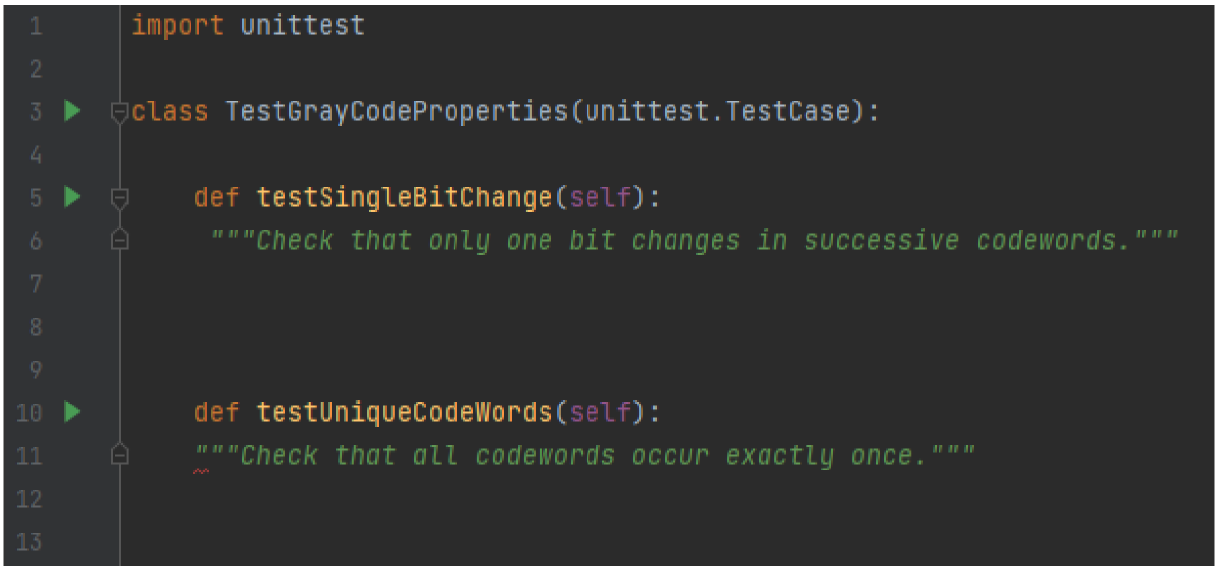
Po uruchomieniu symulacji otrzymano plik testbench.vcd dzięki któremu można zaobserwować przebiegi:



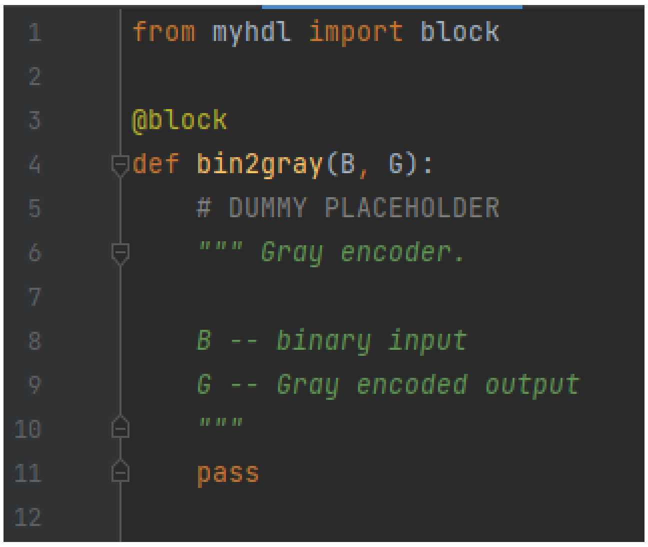
```
Project
14 C:\Users\PC\Desktop\projekty\14
    Include
    Lib
        site-packages library root
            _distutils_hack
            myhdl
            myhdl-0.11-py3.8.egg-info
            pip
                _internal
                _vendor
                __init__.py
                __main__.py
                fsm.py
                py.typed
                testbench.1640200453.6307592.vcd
                testbench.vcd
            pip-21.3.1.dist-info
            pkg_resources
            setuptools
            setuptools-60.0.4.dist-info
            distutils-precedence.pth
    Scripts
    share
    pyvenv.cfg
External Libraries
Scratches and Consoles
```

Unit testing

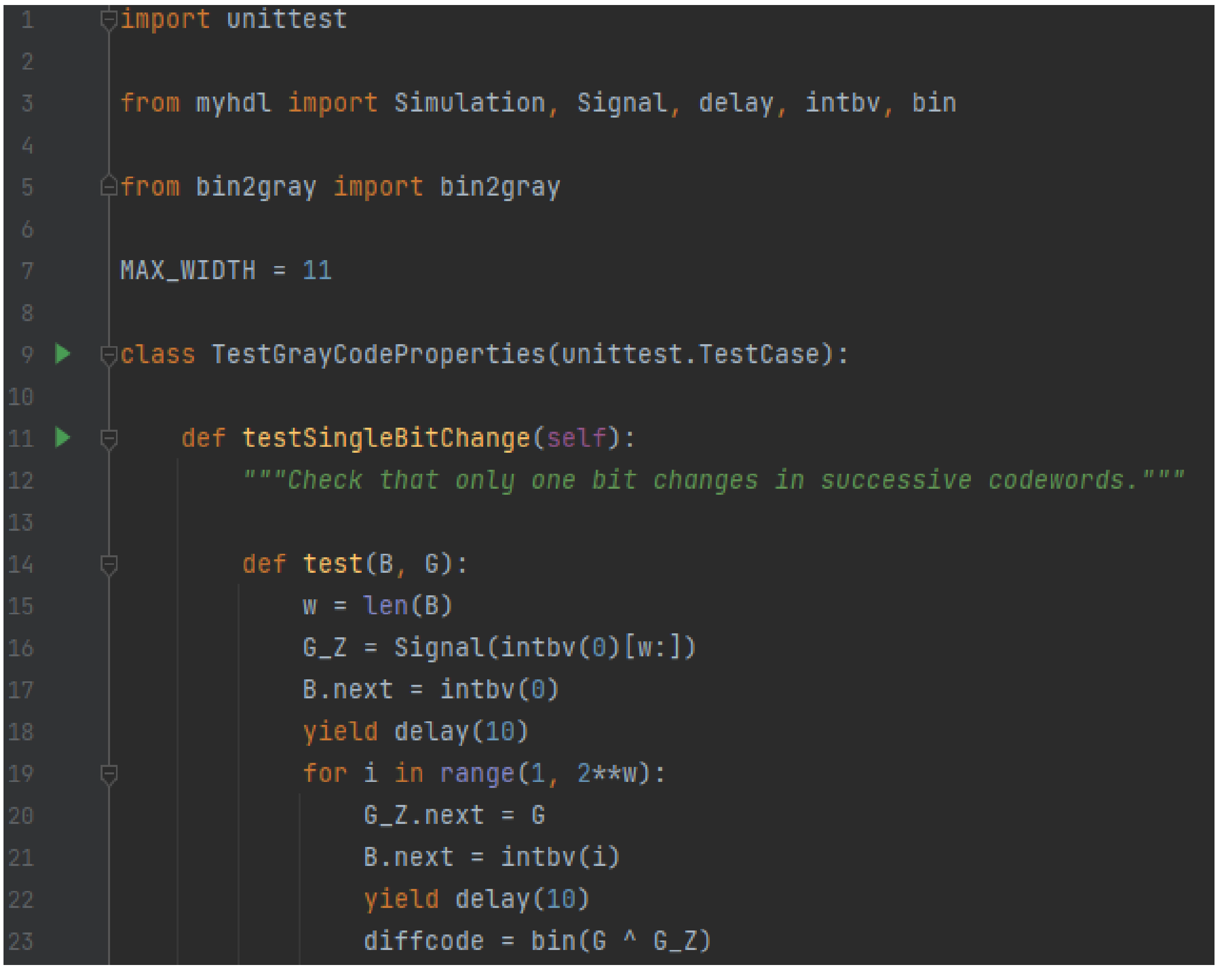
The importance of unit tests



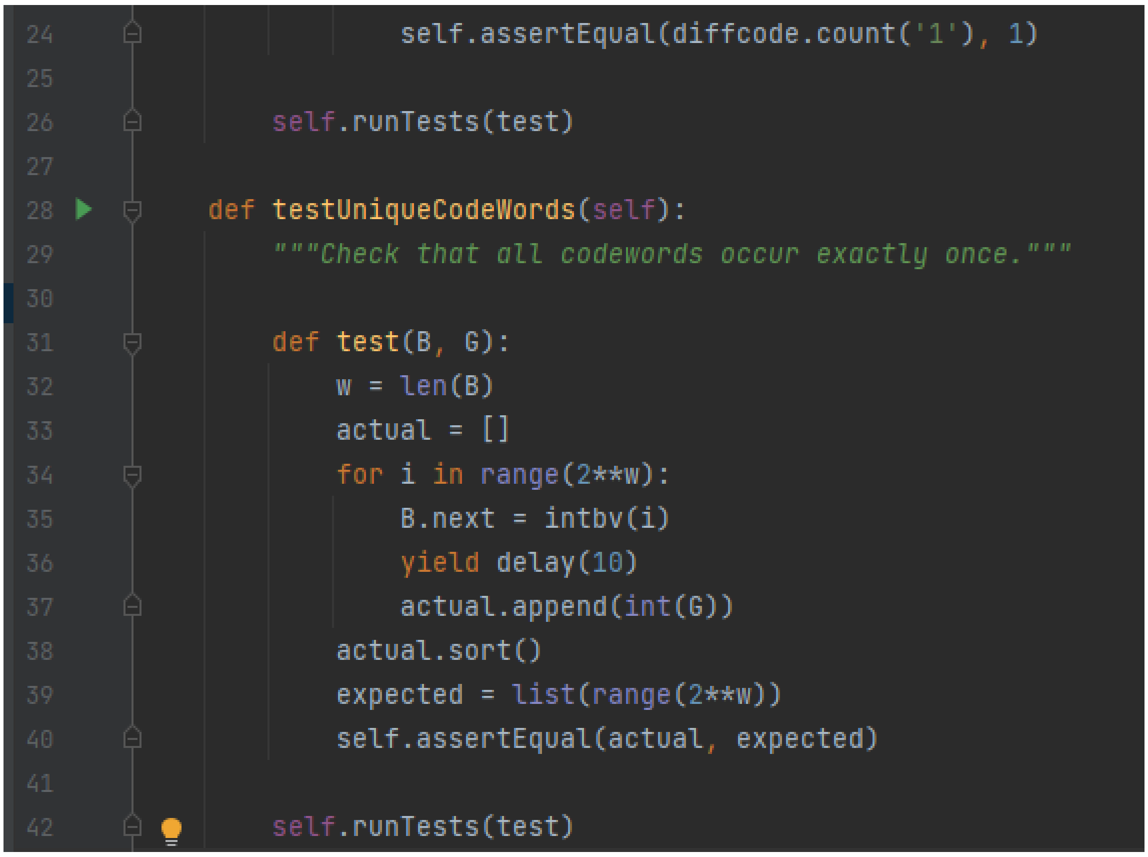
```python
import unittest

class TestGrayCodeProperties(unittest.TestCase):

    def testSingleBitChange(self):
     """Check that only one bit changes in successive codewords."""



    def testUniqueCodeWords(self):
    """Check that all codewords occur exactly once."""


```



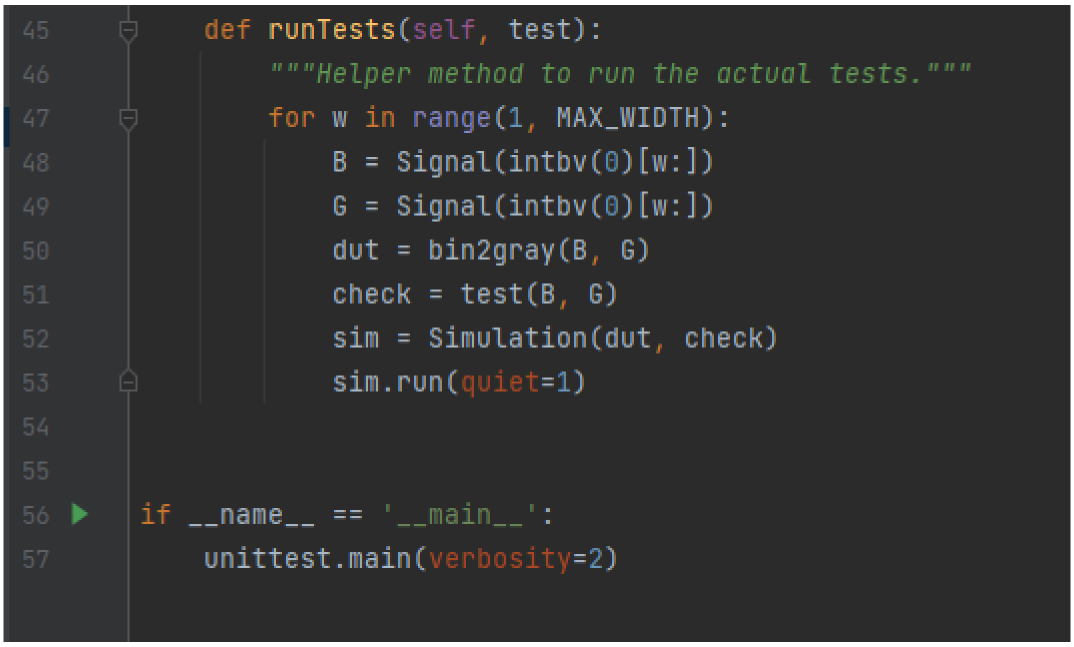
```python
from myhdl import block

@block
def bin2gray(B, G):
    # DUMMY PLACEHOLDER
    """ Gray encoder.

    B -- binary input
    G -- Gray encoded output
    """
    pass

```



```python
import unittest

from myhdl import Simulation, Signal, delay, intbv, bin

from bin2gray import bin2gray

MAX_WIDTH = 11

class TestGrayCodeProperties(unittest.TestCase):

    def testSingleBitChange(self):
        """Check that only one bit changes in successive codewords."""

        def test(B, G):
            w = len(B)
            G_Z = Signal(intbv(0)[w:])
            B.next = intbv(0)
            yield delay(10)
            for i in range(1, 2**w):
                G_Z.next = G
                B.next = intbv(i)
                yield delay(10)
                diffcode = bin(G ^ G_Z)
```

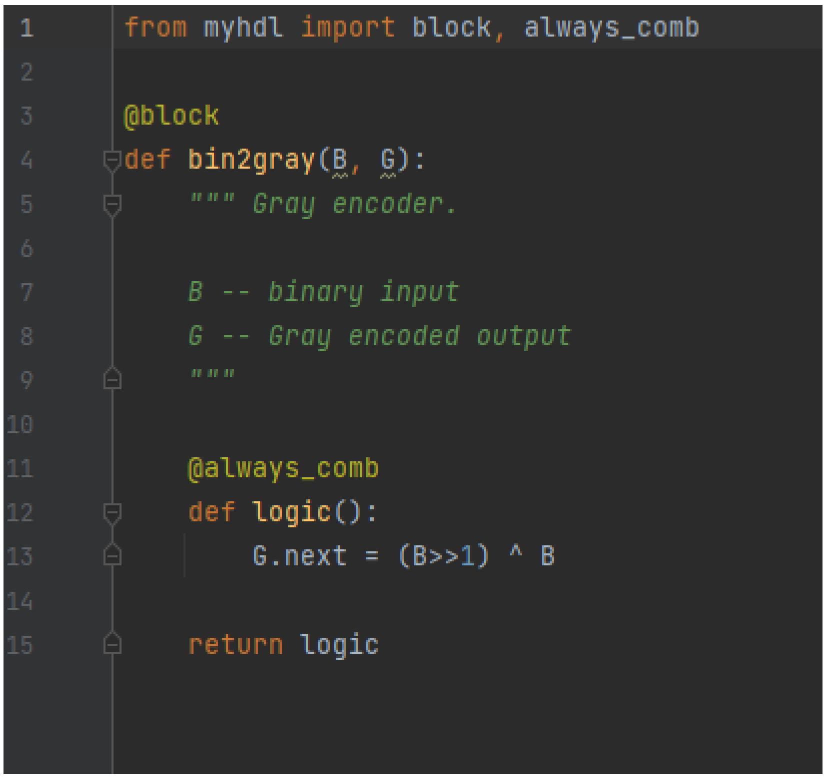


```python
                self.assertEqual(diffcode.count('1'), 1)

        self.runTests(test)

    def testUniqueCodeWords(self):
        """Check that all codewords occur exactly once."""

        def test(B, G):
            w = len(B)
            actual = []
            for i in range(2**w):
                B.next = intbv(i)
                yield delay(10)
                actual.append(int(G))
            actual.sort()
            expected = list(range(2**w))
            self.assertEqual(actual, expected)

        self.runTests(test)
```



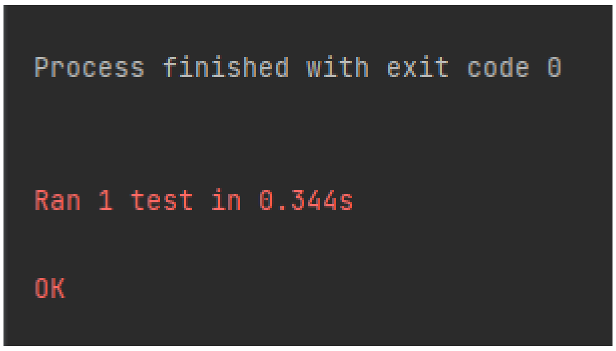
```python
    def runTests(self, test):
        """Helper method to run the actual tests."""
        for w in range(1, MAX_WIDTH):
            B = Signal(intbv(0)[w:])
            G = Signal(intbv(0)[w:])
            dut = bin2gray(B, G)
            check = test(B, G)
            sim = Simulation(dut, check)
            sim.run(quiet=1)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Prawidłowa implementacja „bin2gray”:



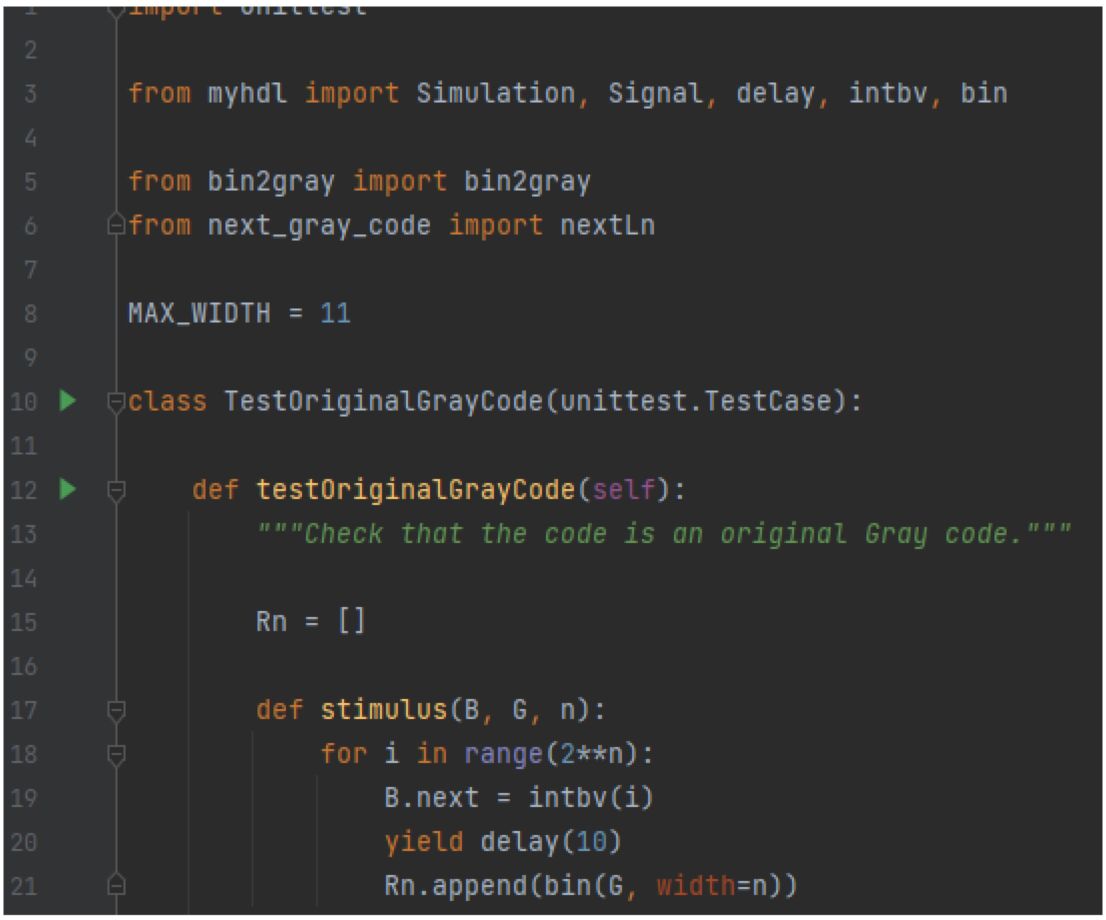
```python
from myhdl import block, always_comb

@block
def bin2gray(B, G):
    """ Gray encoder.

    B -- binary input
    G -- Gray encoded output
    """

    @always_comb
    def logic():
        G.next = (B>>1) ^ B

    return logic
```

Po uruchomieniu symulacji otrzymano:

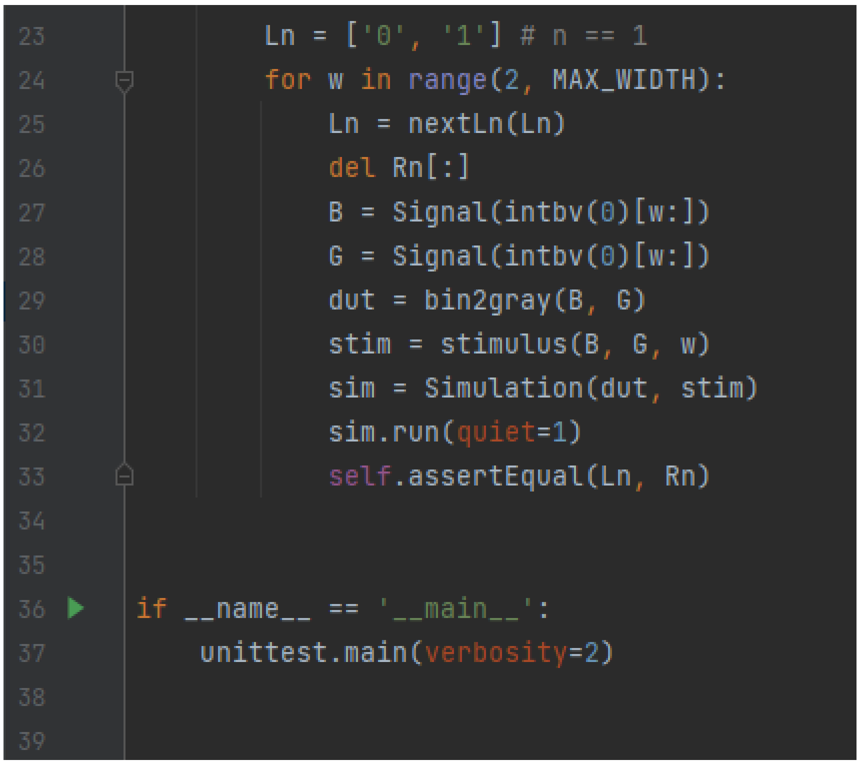


```
Process finished with exit code 0


Ran 1 test in 0.344s

OK
```

nextLn:



```
import unittest

from myhdl import Simulation, Signal, delay, intbv, bin

from bin2gray import bin2gray
from next_gray_code import nextLn

MAX_WIDTH = 11

class TestOriginalGrayCode(unittest.TestCase):

    def testOriginalGrayCode(self):
        """Check that the code is an original Gray code."""

        Rn = []

        def stimulus(B, G, n):
            for i in range(2**n):
                B.next = intbv(i)
                yield delay(10)
                Rn.append(bin(G, width=n))
```



```
        Ln = ['0', '1'] # n == 1
        for w in range(2, MAX_WIDTH):
            Ln = nextLn(Ln)
            del Rn[:]
            B = Signal(intbv(0)[w:])
            G = Signal(intbv(0)[w:])
            dut = bin2gray(B, G)
            stim = stimulus(B, G, w)
            sim = Simulation(dut, stim)
            sim.run(quiet=1)
            self.assertEqual(Ln, Rn)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```